AD705149

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
School of Engineering and Applied Science
Computer Science Department
405 Hilgard, Univ. of Calif., Los Angeles 90024

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE

Computer Network Research

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
ARPA Semiannual Technical Report, August 15, 1969, to February 15, 1970

5. AUTHOR(S) (First name, middle initial, last name)

Leonard Kleinrock

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
February 15, 1970	73	26

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Department of Defense Office of Naval Research

13. ABSTRACT

ARPA Semiannual Technical Report, August 15, 1969, to February 15, 1970

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6801

Security Classification

UNCLASSIFIED

ADVANCED RESEARCH PROJECTS AGENCY

SEMIANNUAL TECHNICAL REPORT

February 15, 1970

Project Computer Network Research ARPA Order Number 1380

Program Code Number P9D30 Contract Number DAHC15-69-C-0285

Effective Date of Contract 4/1/69 Contract Expiration Date 10/31/70*

Amount of Contract $873,109*

Contractor: School of Engineering and Applied Science
Computer Science Department
405 Hilgard Avenue
University of California
Los Angeles, California 90024

Project Scientist and Principal Investigator Dr. Leonard Kleinrock

Phone (213) 825-2543

*Since this contract has had a number of amendments since its initial acceptance, we list below the pertinent chronology:

Date Funds Requested	Period Covered	Amount	Status
November 1968	4/1/69-10/31/69	$229,300	Approved
November 1968	11/1/69-10/31/70	$344,413	"
August 1969	4/1/69-10/31/70	$ 69,396	"
August 1969	11/1/70-6/30/71	$243,345	In process
December 1969	12/1/69-6/30/70	$230,000	" "
December 1969	7/1/70-6/30/71	$300,000	" "

I SUMMARY

The goal of this project is to create an environment suitable for computer research activities in the understanding and in the development of methods for information processing. In particular we will study computer network behavior within the ARPA experimental computer network. Our studies include the mathematical modelling and analysis of the behavior of computer systems emphasizing time-shared computers and computer networks. We also seek to validate the results of such modelling through the use of measurement procedures and will serve as the network measurement center in the ARPA network.

In September, 1969, UCLA became the first node in the ARPA experimental computer network. This took place when we received the special purpose message switching computer (interface message processor-IMP). Within days after the arrival of the IMP, bits and messages were being transferred between the IMP and the UCLA Host computer, the XDS Sigma-7. Since then, the network has grown to include four nodes (UCLA, SRI, UCSB, and the University of Utah); late in October, the first Host-Host messages were transmitted between UCLA and SRI, marking a major milestone in the development of the ARPA computer network.

UCLA, which is to act as the network measurement center has contributed to development of the programs which now function within the IMP for measurement behavior. Simple measurements have already been performed which demonstrate the operation and the usefulness of these measurements techniques. It is expected that this effort will bring about understanding and insight into the network operation and behavior during the next reporting period.

Moreover, it is an ideal tool for validating the mathematical modelling analysis work to which we are devoted.

Progress in the area of mathematical modelling and analysis of computer systems has been significant. A number of papers have been submitted, presented, and published in this area and these are listed as references below. The effort has been directed, mainly, in two areas: time-shared computer systems analysis; and computer network analysis. Research results in the former area have led to the beginning of a comprehensive theory for time-shared scheduling algorithms and their analyses. These results have been submitted and accepted to the highly respected Sixth International Teletraffic Congress to be held in Munich, Germany, September 1970 [Ref. 8]. This work has progressed so well, that it is now time to direct efforts in attempting to model other aspects of time-shared computer operations, such as: memory hierarchy structure; paging effects on the scheduling algorithm; and other congestion points such as input/output congestion. The second area of theoretical results has been in Computer Networks. In the body of this report we include the paper "Analytic and Simulation Methods in Computer Network Design" which has been submitted and accepted for presentation and publication at the forthcoming Spring Joint Computer Conference in Atlantic City, New Jersey. The session at which this paper will be presented is to be devoted entirely to the ARPA Computer Network, and will undoubtedly become the set of papers most referenced with regard to the ARPA Network. The main thrust of this paper is that both analytic and simulation methods have been extremely effective in predicting network behavior and have lead to realistic models of the ARPA network.

In addition to our principal roles as first node in the ARPA network, Network Measurement Center, and Computer Systems Modelling Research Center, we have also been active in establishing the important standards and procedures for carrying out Host-Host communication through appropriate protocol and languages. This effort has been continuing since the earliest discussions of the ARPA Network, and has resulted in a paper "Host-Host Communication Protocol in the ARPA Network" which also has been accepted for presentation and publication at the Spring Joint Computer Conference in the session devoted to the ARPA Network. The contents of that paper are included also in the body of this report following. The concepts put forth in that work represent the results of many people in addition to those of the authors, and it is important to comment that an unusually effective association and interaction has been set up among many network sites, wherein concerned parties have cooperated to create the network protocol described in that paper. This same kind of cooperation has resulted in many benefits to the development of the network, and we feel that the esprit de corps of this ARPA community is an extremely valuable, albeit intangible, asset. The significant aspects of this protocol paper involve design concepts for that protocol within the network. System calls and control commands are defined and suggestions are made regarding the user level languages. A number of problems are solved through the standards set forth in this paper, but as is to be expected, many more are created which as yet need resolution from among the users of this network. Clearly growth in this area must continue and will follow with this paper as the major point of departure.

The response from the computer community to the activities taking place at UCLA has been more than encouraging. We have achieved recognition as one

of the leading computer systems modelling and analyses centers in the world. Our efforts in measurement of the network behavior have stirred up considerable interest. The impact of the partial solutions offered to the Host-Host Protocol and Language problem are just now beginning to be felt.

II TECHNICAL REPORT

Among the numerous areas of investigation carried out during this reporting period, we choose to elaborate upon two in this report. The first dwells on analytic and simulation models suitable for computer network design, as mentioned in the summary. This paper follows as Section II.1 and is made up of the paper submitted to the Spring Joint Computer Conference.

The second effort emphasized here is that of the HOST-HOST Communication Protocol in the ARPA Network, and is presented as Section II.2, also in the form of the submitted paper for the Spring Joint Computer Conference.

Each paper contains its own reference list and is thereby self-contained.

PUBLICATIONS SUPPORTED UNDER ARPA CONTRACT #DAHC15-69-C-0285

1. Carr, C. S., S. D. Crocker, and V. G. Cerf, "HOST-HOST Communication Protocol in the ARPA Network," to be presented at and published in Proc. of the SJCC, Atlantic City, N.J., May 1970.

2. Chu, W. W., "Selection of Optimal Transmission Rate for Statistical Multiplexors," to be presented at and published in the Proc. of the IEEE International Conference on Communications, San Francisco, June 8-10, 1970.

3. Coffman, E. G., Jr., and R. R. Muntz, "Model of Pure Time Sharing Disciplines for Resource Allocation," Proc. of the 24th National Conference of ACM, August 1969.

4. Kleinrock, L., "Swap Time Considerations in Time-Shared Systems," IEEE Transactions on Computers, August 1969.

5. Kleinrock, L., "Comparison of Solution Methods for Computer Network Models," Proc. of the Computer and Communications Conference, Rome, N.Y., Oct. 2, 1969.

6. Kleinrock, L., "A Continuum of Time-Sharing Scheduling Algorithms," to be presented at and published in Proc. of the SJCC, Atlantic City, N.J., May 1970.

7. Kleinrock, L., "Analytic and Simulation Methods in Computer Network Design," to be presented and published in Proc. of the SJCC, Atlantic City, N.J., May 1970.

8. Kleinrock, L., and R. R. Muntz, "Multilevel Processor-Sharing Queueing Models for Time-Shared Systems," to be presented at and published in Proc. of the Sixth International Teletraffic Congress, Munich, Germany, August 1970.

9. Muntz, R. R., and R. Uzgalis, "Dynamic Storage Allocation for Binary Search Trees in a Two-Level Memory," Proc. of the Fourth Annual Princeton Conference on Information Sciences and Systems, Princeton, N.J., March 26-27, 1970.

PUBLICATIONS OF INTEREST SUPPORTED
UNDER THE PREVIOUS ARPA CONTRACT #SD-184

10. Baer, J., and G. Estrin, "Frequency Numbers Associated with Directed Graph Representations of Computer Programs," Second Hawaii International Conference on System Sciences, Honolulu, Hawaii, January 22-24, 1969.

11. Coffman, G., and L. Kleinrock, "Some Feedback Queueing Models for Time-Shared Systems," Proc. of the Fifth International Teletraffic Congress, New York, pp. 288-304, June 13-19, 1967.

12. Estrin, G., and L. Kleinrock, "Measures, Models and Measurements for Time-Shared Computer Utilities," Proc. of the 22nd ACM National Meeting, Washington, D.C. pp. 85-96, August 1967.

13. Kleinrock, L., "Time-Shared Systems: A Theoretical Treatment," JACM, Vol. 14, pp. 242-261, 1967.

14. Kleinrock, L., "Distribution of Attained Service in Time-Shared Systems," Journal of Computers and System Science, Vol. 1, No. 3, pp. 287-298, October 1967.

15. Kleinrock, L., "Some Recent Results for Time-Shared Processors," Proc. of the International Conference on System Sciences, University of Hawaii, Honolulu, January 29-31, 1968.

16. Kleinrock, L., and E. G. Coffman, "Computer Scheduling Methods and their Countermeasures," Proc. of the Spring Joint Computer Conference, Atlantic City, N.J., pp. 11-21, April 30-May 2, 1968.

17. Kleinrock, L., "Some Results on the Design of Communication Nets," Proc. of the IEEE International Communications Conference. Philadelphia, Pa., pp. 699-705, June 12-14, 1968.

18. Kleinrock, L., "Certain Analytical Results for Time-Shared Processors," Proc. of the IFIP Congress, Edinburgh, Scotland, pp. D119-D125, August 5-10, 1968.

19. Kleinrock, L., and E. G. Coffman, "Feedback Queueing Models for Time-Shared Systems," Journal of the ACM, Vol. 15, No. 4, pp. 549-576, October 1968.

20. Kleinrock, L., "Time-Sharing Systems: Analytical Methods," Proc. of the Symposium on Critical Factors in Data Management/1968, UCLA, March 20-22, 1968, published by Prentice-Hall, pp. 3-32, 1969.

21. Kleinrock, L., "Models for Computer Networks," Proc. of the IEEE International Conference on Communications, Boulder, Colo., pp. 21-16 to 21-29, June 9-11, 1969.

22. Kleinrock, L., "On Swap Time in Time-Shared Systems," Proc. of the IEEE Computer Group Conference, Minneapolis, Minn., pp. 37-41, June 17-19, 1969.

23. Martin, D., and G. Estrin, "Models of Computational Systems-Cyclic to Acyclic Graph Transformations," IEEE Transactions on Electronic Computers, Vol, EC-16, pp. 70-79, February 1967.

24. Martin, D., and G. Estrin, "Experiments on Models of Computations and Systems," IEEE Transactions on Electronic Computers, Vol. EC-16, pp. 59-69, February 1967.

25. Martin, D., and G. Estrin, "Models of Computations and Systems-Evaluation of Vertex Probabilities in Graph Models of Computations," Journal of ACM, Vol. 2, No. 14, pp. 281-299, April 1967.

26. Martin, D., and G. Estrin, "Path Length Computations on Graph Models of Computations," Transactions of the IEEE, Vol. C-18, pp. 530-536, June 1969.

II.1 Analytic and Simulation Methods in Computer Network Design*

(This is a paper to be presented at SJCC '70 written by

Leonard Kleinrock

Computer Science Department

University of California at Los Angeles

Los Angeles, California 90024)

*This work was supported by the Advanced Research Projects Agency of the Department of Defense # DAHC15-69-C-0285.

ANALYTIC AND SIMULATION METHODS IN COMPUTER NETWORK DESIGN*

Leonard Kleinrock
Computer Science Department
University of California at Los Angeles
Los Angeles, California 90024

(213) 825-2543

ABSTRACT

This paper addresses itself to problems and solutions in the mathematical analysis and simulation of computer networks. A framework is constructed around which a useful theory of computer networks can be developed. The results so far obtained provide meaningful insight and useful aids in analysis and design of these systems. The ARPA experimental computer network is used as an example against which the methods of this paper can be compared.

The paper divides in three parts. The first creates a mathematical queueing model which is then analyzed to yield the average message delay for messages travelling through the network. These analytic computations are then compared to simulation results for the ARPA computer network in a given configuration; a model is found for which the agreement between theory and simulation is amazingly good. The second part addresses itself to the synthesis and optimization question; this requires the definition of an appropriate cost function for the network and we carefully examine a variety of such cost functions which resemble available data on commercial transmission systems.

---

*This work was supported by the Advanced Research Projects Agency of the Department of Defense (DAH15-69-C-0285).

The optimization then reduces to finding that distribution of channel capacity within the network which minimizes the average message delay at a fixed network cost. These optimal designs are then compared on the basis of average message delay, system cost, and throughput data rate. This comparison shows that the particularly simple linear cost function (which is well understood and easy to solve) approximates a much more complicated (albeit more realistic) cost function, namely the power law cost function. The fact that the power law case can be well approximated by the linear case is most valuable since the linear case yields completely to analytic methods in solving for the optimal distribution of capacity in networks. The third part considers some aspects of the operating procedure within a computer network. In particular, the important question of how one should modify and update the network routing procedure is considered. It is shown from simulation that for the ARPA network an asynchronous method for updating is superior to the synchronous method in that it provides smaller average message delays; however, the cost for asynchronous updating has yet to be accounted for and the software overhead for this method must be studied in terms of its effect on message delay and throughput. It is also shown that the synchronous updating method includes transient looping effects which if removed can provide reduced message delays as well.

The results obtained so far are most encouraging and it is vital that these methods be extended to consider other performance measures and network parameters so as to sharpen these already useful tools.

ANALYTIC AND SIMULATION METHODS IN COMPUTER NETWORK DESIGN *

by
Leonard Kleinrock

INTRODUCTION

The Seventies are here and so are computer networks! The time sharing industry dominated the Sixties and it appears that computer networks will play a similar role in the Seventies. The need has now arisen for many of these time shared systems to share each others' resources by coupling them together over a communication network thereby creating a computer network. The mini-computer will serve an important role here as the sophisticated terminal as well as, perhaps, the message switching computer in our networks.

It is fair to say that the computer industry (as is true of most other large industries in their early development) has been guilty of "leaping before looking"; on the other hand "losses due to hesitation" are not especially prevalent in this industry. In any case, it is clear that much is to be gained by an appropriate mathematical analysis of performance and cost measures for these large systems, and that these analyses should most profitably be undertaken before major design commitments are made. This paper attempts to move in the direction of providing some tools for and insight into the design of computer networks through mathematical modeling, analysis and simulation. Frank, et al.[4] describe tools for obtaining low cost networks by choosing among topologies using computationally efficient methods from network flow theory; our approach complements theirs in that we

* This work was supported by the Advanced Research Projects Agency of the Department of Defense (DAHC15-69-C-0285).

look for closed analytic expressions where possible. Our intent is to provide understanding of the behavior and trade-offs available in some computer network situations thus creating a qualitative tool for choosing design options and not a numerical tool for choosing precise design parameters.

THE ARPA EXPERIMENTAL COMPUTER NETWORK - AN EXAMPLE

The particular network which we shall use for purposes of example (and with which we are most familiar) is the Defense Department's Advanced Research Projects Agency (ARPA) experimental computer network.[2] The concepts basic to this network were clearly stated in Reference 11 by L. Roberts of the Advanced Research Projects Agency, who originally conceived this system. Reference 6, which appears in these proceedings, provides a description of the historical development as well as the structural organization and implementation of the ARPA network. We choose to review some of that description below in order to provide the reader with the motivation and understanding necessary for maintaining a certain degree of self containment in this paper.

As might be expected, the design specifications and configuration of the ARPA network have changed many times since its inception in 1967. In June, 1969, this author published a paper[8] in which a particular network configuration was described and for which certain analytical models were constructed and studied. That network consisted of nineteen nodes in the continental United States. Since then this number has changed and the identity of the nodes has changed and the topology has changed, and so on. The paper by Frank, et al.,[4] published in these proceedings, describes the behavior and topological design of one of these newer versions. However, in order to be consistent with our earlier results, and since the ARPA example is intended

as an illustration of an approach rather than a precise design computation, we choose to continue to study and therefore to describe the original nineteen node network in this paper.

The network provides store-and-forward communication paths between the set of nineteen computer research centers. The computers located at the various nodes are drawn from a variety of manufacturers and are highly incompatible both in hardware and software; this in fact presents the challenge of the network experiment, namely, to provide effective communication among and utilization of this collection of incompatible machines. The purpose is fundamentally for resource sharing where the resources themselves are highly specialized and take the form of unique hardware, programs, data bases, and human talent. For example, Stanford Research Institute will serve the function of network librarian as well as provide an efficient text editing system; the University of Utah provides efficient algorithms for the manipulation of figures and for picture processing; the University of Illinois will provide through its ILLIAC IV the power of its fantastic parallel processing capability; UCLA will serve as network measurement center and also provide mathematical models and simulation capability for network and time-shared system studies.

The example set of nineteen nodes is shown in Figure 1 (as of Spring 1969). The traffic matrix which describes the message flow required between various pairs of nodes is given in Reference 8 and will not be repeated here. An underlying constraint placed upon the construction of this network was that network operating procedures would not interfere in any significant way with the operation of the already existing facilities which were to be connected together through this network. Consequently the message handling tasks (relay, acknowledgment, routing, buffering, etc.) are carried out in a special purpose Interface Message Processor (IMP) co-located

with the principal computer (denoted HOST computer) at each of the computer research centers. The communication channels are (in most cases) 50 kilobit per second full duplex telephone lines and only the IMPs are connected to these lines through data sets. Thus the communication net consists of the lines, the IMPs and the data sets and serves as the store-and-forward system for the HOST computer network. Messages which flow between HOST's are broken up into small entities referred to as packets (each of maximum size of approximately 1000 bits). The IMP accepts up to eight of these packets to create a maximum size message from the HOST. The packets make their way individually through the IMP network where the appropriate routing procedure directs the traffic flow. A positive acknowledgment is expected within a given time period for each inter-IMP packet transmission; the absence of an acknowledgment forces the transmitting IMP to repeat the transmission (perhaps over the same channel or some other alternate channel). An acknowledgment may not be returned for example, in the case of detected errors or for lack of buffer space in the receiving IMP. We estimate the average packet size to be 560 bits; the acknowledgment length is assumed to be 140 bits. Thus, if we assume that each packet transmitted over a channel causes the generation of a positive acknowledgment packet (the usual case, hopefully), then the average packet transmission over a line is of size 350 bits. Much of the short interactive traffic is of this nature. We also anticipate message traffic of much longer duration and we refer to this as multi-packet traffic. The average input data rate to the entire net is assumed to be 225 kilobits per second and again the reader is referred to Reference 8 for further details of this traffic distribution.

So much for the description of the ARPA network. Protocol and operating

procedures for the ARPA computer network are described in References 1 and 6 in these proceedings in much greater detail. The history, development, motivation and cost of this network is described by its originator in Reference 12. Let us now proceed to the mathematical modelling, analysis and simulation of such networks.

ANALYTIC AND SIMULATION METHODS

The mathematical tools for computer network design are currently in the early stages of development. In many ways we are still at the stage of attempting to create computer network models which contain enough salient features of the network so that behavior of such networks may be predicted from the model behavior.

In this section we begin with the problem of analysis for a given network structure. First we review the author's earlier analytic model of communication networks and then proceed to identify those features which distinguish computer networks from strict communication networks. Some previously published results on computer networks are reviewed and then new improvements on these results are presented.

We then consider the synthesis and optimization question for networks. We proceed by first discussing the nature of the channel cost function as available under present tariff and charging structures. We consider a number of different cost functions which attempt to approximate the true data and derive relationships for optimizing the selection of channel capacities under these various cost functions. Comparisons among the optimal solutions are then made for the ARPA network.

Finally in this section we consider the operating rules for computer networks. We present the results of simulation for the ARPA network regarding certain aspects of the routing procedure which provide improvements in performance.

A Model from Queueing Theory - Analysis

In a recent work[8] this author presented some computer network models which were derived from his earlier research on communication networks[7]. An attempt was made at that time to incorporate many of the salient features of the ARPA network described above into this computer network model. It was pointed out that computer networks differ from communication networks as studied in Reference 7 in at least the following features: (a) nodal storage capacity is finite and may be expected to fill occasionally; (b) channel and modem errors occur and cause re-transmission; (c) acknowledgment messages increase the message traffic rates; (d) messages from HOST A to HOST B typically create return traffic (after some delay) from B to A; (e) nodal delays become important and comparable to channel transmission delays; (f) channel cost functions are more complex. We intend to include some of these features in our model below.

The model proposed for computer networks is drawn from our communication network experience and includes the following assumptions. We assume that the message arrivals form a Poisson process with average rates taken from a given traffic matrix (such as in Reference 8), where the message lengths are exponentially distributed with a mean $1/\mu$ of 350 bits (note that we are only accounting for short messages and neglecting the multi-packet traffic in this model). As discussed at length in Reference 7, we also make the

independence assumption which allows a very simple node by node analysis. We further assume that a fixed routing procedure exists (that is, a unique allowable path exists from origin to destination for each origin-destination pair).

From the above assumptions one may calculate the average delay T_i due to waiting for and transmitting over the i^{th} channel from Eq. (1),

$$T_i = \frac{1}{\mu C_i - \lambda_i} \tag{1}$$

where λ_i is the average number of messages per second flowing over channel i (whose capacity is C_i bits per second). This was the appropriate expression for the average channel delay in the study of communication nets[7] and in that study we chose as our major performance measure the message delay T averaged over the entire network as calculated from

$$T = \sum_i \frac{\lambda_i}{\gamma} T_i \tag{2}$$

where γ equals the total input data rate. Note that the average on T_i is formed by weighting the delay on channel C_i with the traffic, λ_i, carried on that channel. In the study of communication nets[7] this last equation provided an excellent means for calculating the average message delay. That study went on to optimize the selection of channel capacity throughout the network under the constraint of a fixed cost which was assumed to be linear with capacity; we elaborate upon this cost function later in this section.

The computer network models studied in Reference 8 also made use of Eq. (1) for the calculation of the channel delays (including queueing) where parameter choices were $1/\mu$ = 350 bits, C_i = 50 kilobits and λ_i = average message rate on channel i (as determined from the traffic matrix, the routing procedure, and accounting for the effect of acknowledgment traffic as mentioned in feature (c) above). In order to account for feature (e) above, the performance measure (taken as the average message delay T) was calculated from

$$T = \sum_i \frac{\lambda_i}{\gamma} (T_i + 10^{-3}) \tag{3}$$

where again γ = total input data rate and the term 10^{-3} = 1 millisecond (nominal) is included to account for the assumed (fixed) nodal processing time. The result of this calculation for the ARPA network shown in Figure 1 may be found in Reference 8.

The computer network model described above is essentially the one used for calculating delays in the topological studies reported upon by Frank et al. in these proceedings.[4]

A number of simulation experiments have been carried out using a rather detailed description of the ARPA network and its operating procedure. Some of these results were reported upon in Reference 8 and a comparison was made there between the theoretical results obtained from Eq. (3) and the simulation results. This comparison is reproduced in Figure 2 where the lowest curve corresponds to the results of Eq. (3). Clearly the comparison between simulation and theory is only mildly satisfactory. As pointed out in Reference 8,

the discrepancy is due to the fact that the acknowledgment traffic has been improperly included in Equation 3. An attempt was made in Reference 8 to properly account for the acknowledgment traffic; however, this adjustment was unsatisfactory. The problem is that the average message length has been taken to be 350 bits and this length has averaged the traffic due to acknowledgment messages along with traffic due to real messages. These acknowledgments should not be included among those messages whose average system delay is being calculated and yet acknowledgment traffic must be included to properly account for the true loading effect in the network. In fact, the appropriate way to include this effect is to recognize that the time spent waiting for a channel is dependent upon the total traffic (including acknowledgments) whereas the time spent in transmission over a channel should be proportional to the message length of the real message traffic. Moreover, our theoretical equations have accounted only for transmission delays which come about due to the finite rate at which bits may be fed into the channel (i.e., 50 kilobits per second); we are required however to include also the propagation time for a bit to travel down the length of the channel. Lastly, an additional one millisecond delay is included in the final destination node in order to deliver the message to the destination HOST. These additional effects give rise to the following expression for the average message delay T.

$$T = \sum_i \frac{\lambda_i}{\gamma} \left(\frac{1}{\mu' C_i} + \frac{\lambda_i / \mu C_i}{\mu C_i - \lambda_i} + PL_i + 10^{-3} \right) + 10^{-3} \qquad (4)$$

where $1/\mu' = 560$ bits (a real message's average length) and PL_i is the propagation delay (dependent on the channel length, L_i) for the i^{th} channel. The

first term in parentheses is the average transmission time and the second term is the average waiting time. The result of this calculation for the ARPA network gives us the curve in Figure 2 labelled "theory with correct acknowledge adjustment and propagation delays." The correspondence now between simulation and theory is unbelievably good and we are encouraged that this approach appears to be a suitable one for the prediction of computer network performance for the assumptions made here. In fact, one can go further and include the effect on message delay of the priority given to acknowledgment traffic in the ARPA network; if one includes this effect, one obtains another excellent fit to the simulation data labelled in Figure 2 as "theory corrected and with priorities."

As discussed in Reference 8 one may generalize the model considered herein to account for more general message length distributions by making use of the Pollaczek-Khinchin formula for the delay T_i of a channel with capacity C_i, where the message lengths have mean $1/\mu$ bits with variance σ^2, where λ_i is the average message traffic rate and $\rho_i = \lambda_i/\mu C_i$ which states

$$T_i = \frac{1}{\mu' C_i} + \frac{\rho_i(1 + \mu^2\sigma^2)}{2(\mu C_i - \lambda_i)} \qquad (5)$$

This expression would replace the first two terms in the parenthetical expression of Eq. (4); of course by relaxing the assumption of an exponential distribution we remove the simplicity provided by the Markovian property of the traffic flow. This approach, however, should provide a better approximation to the true behavior when required.

Having briefly considered the problem of analyzing computer networks with regard to a single performance measure (average message delay), we now move on to the consideration of synthesis questions. This investigation immediately leads into optimal synthesis procedures.

Optimization for Various Channel Cost Functions--Synthesis

We are concerned here with the optimization of the channel capacity assignment under various assumptions regarding the cost of these channels. This optimization must be made under the constraint of fixed cost. Our problem statement then becomes:*

$$\text{Select the } \{C_i\} \text{ so as to minimize } T \quad \text{subject to a fixed cost constraint} \tag{6}$$

where, for simplicity, we use the expression in Eq. (2) to define T.

We are now faced with choosing an appropriate cost function for the system of channels. We assume that the total cost of the network is contained in these channel costs where we certainly permit fixed termination charges, for example, to be included. In order to get a feeling for the correct form for the cost function let us examine some available data. From Reference 3 we have available the costing data which we present in Table 1. From a schedule of costs for leased communication lines available to ARPA we have the data presented in Table 2.

---

*The dual to this optimization problem may also be considered: "Select the $\{C_i\}$ so as to minimize cost, D, subject to a fixed message delay constraint." The solution to this dual problem gives the optimum C_i with the same functional dependence on λ_i as one obtains for the original optimization problem.

TABLE I--Publicly available leased transmission line costs from Reference 3

Speed	Cost/mile/month (normalized to 1000 mile distance)
45 bps	$ .70
56 bps	.70
75 bps	.77
2400 bps	1.79
41 KB	15.00
82 KB	20.00
230 KB	28.00
1 MB	60.00
12 MB	287.50

TABLE II--Estimated leased transmission line costs based on Telpak rates*

Speed	Cost (termination + mileage) /month	Cost/mile/month (normalized to 1000 mile distance)
150 bps	$ 77.50 + $ .12/mile	$ .20
2400 bps	232. + .35/mile	.58
7200 bps	810 + .35/mile	1.16
19.2 KB	850 + 2.10/mile	2.95
50 KB	850 + 4.20/mile	5.05
108 KB	2400 + 4.20/mile	6.60
230.4 KB	1300 + 21.00/mile	22.30
460.8 KB	1300 + 60.00/mile	61.30
1.344 MB	500 + 75.00/mile	80.00

*These costs are, in some cases, first estimates and are not to be considered as quoted rates.

We have plotted these functions in Figure 3. We must now attempt to find an analytic function which fits cost functions of this sort. Clearly that analytic function will depend upon the rate schedule available to the computer network designer and user. Many analytic fits to this function have been proposed and in particular in References 3 a fit is proposed of the form:

$$\text{Cost of line} = 0.1\ C_i^{0.44} \quad \$/\text{mile/month} \qquad (7)$$

Based upon rates available for private line channels, Mastromonaco[10] arrives at the following fit for line costs where he has normalized to a distance of 50 miles (rather than 1000 miles in Eq. (7))

$$\text{Cost of line} = 1.08\ C_i^{0.316} \quad \$/\text{mile/month} \qquad (8)$$

Referring now to Figure 3 we see that the mileage costs from Table II rise as a fractional exponent of capacity (in fact with an exponent of .815) suggesting the cost function shown in Eq. (9) below

$$\text{Cost of line} = A\ C_i^{.815} \quad \$/\text{mile/month} \qquad (9)$$

These last three equations give the dollar cost per mile per month where the capacity C_i is given in bits per second. It is interesting to note that all three functions are of the form

$$\text{Cost of line} = A\ C_i^{\alpha} \quad \$/\text{mile/month} \qquad (10)$$

It is clear from these simple considerations that the cost function appropriate for a particular application depends upon that application and therefore it is difficult to establish a unique cost function for all situations. Consequently,

we satisfy ourselves below by considering a number of possible cost functions and study optimization conditions and results which follow from those cost functions. The designer may then choose from among these to match his given tariff schedule. These cost functions will form the fixed cost constraint in Eq. (6). Let us now consider the collection of cost functions, and the related optimization questions.

1. Linear cost function. We begin with this case since the analysis already exists in the author's Reference 7, where the assumed cost constraint took the form

$$D = \sum_i d_i C_i \tag{11}$$

where D = total number of dollars available to spend on channels, d_i = the dollar cost per unit of capacity on the i^{th} channel, and C_i once again is the capacity of the i^{th} channel. Clearly Eq. (11) is of the same form as Eq. (10) with $\alpha = 1$ where we now consider the cost of all channels in the system as having a linear form. This cost function assumes that cost is strictly linear with respect to capacity; of course this same cost function allows the assumption of a constant (for example, termination charges) plus a linear cost function of capacity. This constant (termination charge) for each channel may be subtracted out of total cost, D, to create an equivalent problem of the form given in Eq. (11). The constant, d_i, allows one to account for the length of the channel since d_i may clearly be proportional to the length of the channel as well as anything else regarding the particular channel involved such as, for example, the terrain over which the channel must be placed. As was done in Reference 7, one may carry out the minimization given by Eq. (6) using, for

example, the method of Lagrangian undetermined multipliers.[5] This procedure yields the following equation for the capacity

$$C_i = \frac{\lambda_i}{\mu} + \left(\frac{D_e}{d_i}\right) \frac{\sqrt{\lambda_i d_i}}{\sum_j \sqrt{\lambda_j d_j}} \tag{12}$$

where

$$D_e = D - \sum_i \frac{\lambda_i d_i}{\mu} > 0 \tag{13}$$

When we substitute this result back into Eq. (2) we obtain that the performance measure for such a channel capacity assignment is

$$T = \frac{\bar{n}\left(\sum_i \sqrt{\lambda_i d_i / \lambda}\right)^2}{\mu D_e} \tag{14}$$

where

$$\bar{n} = \frac{\sum_i \lambda_i}{\gamma} \equiv \frac{\lambda}{\gamma} = \text{average path length} \tag{15}$$

The resulting Eq. (12) is referred to as the square root channel capacity assignment; this particular assignment first provides to each channel a capacity equal to λ_i/μ which is merely the average bit rate which must pass over that channel and which it must obviously be provided if the channel is to carry such traffic. In addition, surplus capacity (due to excess dollars, D_e) is assigned to this channel in proportion to the square root of the traffic carried, hence the name. In Reference 7 the author studied in great detail the particular case for which $d_i = 1$ (the case for which all channels cost the same regardless of length) and considerable information regarding topological

design and routing procedures was thereby obtained. However, in the case of the ARPA network a more reasonable choice for d_i is that it should be proportional to the length L_i of the i^{th} channel as indicated in Eq. (10) (for $\alpha = 1$) which gives the per mileage cost; thus we may take $d_i = AL_i$. This second case was considered in Reference 8 and also in Reference 9. The interpretation for these two cases regarding the desirability of concentrating traffic into a few large and short channels as well as minimizing the average length of lines traversed by a message was well discussed and will not be repeated here.

We observe in the ARPA network example since the channel capacities are fixed at 50 kilobits that there is no freedom left to optimize the choice of channel capacities; however it was shown in Reference 8 that one could take advantage of the optimization procedure in the following way: The total cost of the network using 50 kilobit channels may be calculated. One may then optimize the network (in the sense of minimizing T) by allowing the channel capacities to vary while maintaining the cost fixed at this figure. The result of such optimization will provide a set of channel capacities which vary considerably from the fixed capacity network. It was shown in Reference 8 that one could improve the performance of the network in an efficient way by allowing that channel which required the largest capacity as a result of optimization to be increased from 50 kilobits in the fixed net to 250 kilobits. This of course increases the cost of the system. One may then provide a 250 kilobit channel for the second "most needy" channel from the optimization, increasing the cost further. One may then continue this procedure of increasing the needy channels to 250 kilobits while increasing the cost of the network and observe the way in which message delay decreases as system cost increases. It was

found that natural stopping points for this procedure existed at which the cost increased rapidly without a similar sharp decrease in message delay thereby providing some handle on the cost-performance trade-off.

Since we are more interested in the difference between results obtained when one varies the cost function in more significant ways, we now study additional cost functions.

2. Logarithmic cost functions. The next case of interest assumes a cost function of the form

$$D = \sum_i d_i \log_e \alpha C_i \tag{16}$$

where D again is the total dollar cost provided for constructing the network, d_i is a coefficient of cost which may depend upon length of channel, α is an appropriate multiplier and C_i is the capacity of the i^{th} channel. We consider this cost function for two reasons: first, because it has the property that the incremental cost per bit decreases as the channel size increases; and secondly, because it leads to simple theoretical results. We now solve the minimization problem expressed in Eq. (6) where the fixed cost constraint is now given through Eq. (16). We obtain the following equation for the capacity of the i^{th} channel.

$$C_i = \frac{\lambda_i}{\mu}\left[1 + \frac{1}{2\gamma\beta d_i} + \left(\frac{1}{\gamma\beta d_i} + \left(\frac{1}{2\gamma\beta d_i}\right)^2\right)^{1/2}\right] \tag{17}$$

In this solution the Lagrangian multiplier β must be adjusted so that Eq. (16) is satisfied when C_i is substituted in from Eq. (17). Note the unusual simplicity for the solution of C_i, namely that the channel capacity for the i^{th}

channel is directly proportional to the traffic carried by that channel, λ_i/μ. Contrast this result with the result in Eq. (12) where we had a square root channel capacity assignment. If we now take the simple result given in Eq. (17) and use it in Eq. (2) to find the performance measure T we obtain

$$T = \sum_i \frac{1}{\frac{1}{2d_i\beta} + \left[\frac{\gamma}{d_i\beta} + \left(\frac{1}{2d_i\beta}\right)^2\right]^{1/2}} \tag{18}$$

In this last result the performance measure depends upon the particular distribution of the internal traffic $\{\lambda_i/\mu\}$ through the constant β which is adjusted as described above.

3. The power law cost function. As we saw in Eqs. (7), (8), and (9) it appears that many of the existing tariffs may be approximated by a cost function of the form given in Eq. (19) below.

$$D = \sum_i d_i C_i^{\alpha} \tag{19}$$

where α is some appropriate exponent of the capacity and d_i is an arbitrary multiplier which may of course depend upon the length of the channel and other pertinent channel parameters. Applying the Lagrangian again with an undetermined multiplier β we obtain as our condition for an optimal channel capacity the following non-linear equation:

$$C_i - \frac{\lambda_i}{\mu} - C_i^{\frac{1-\alpha}{2}} g_i = 0 \tag{20}$$

where

$$g_i = \left(\frac{\lambda_i}{\mu\gamma\beta\alpha d_i}\right)^{1/2} \tag{21}$$

Once again, β must be adjusted so as to satisfy the constraint Eq. (19).

It can be shown that the left hand side of Eq. (20) represents a convex function and that it has a unique solution for some positive value C_i. We assume that α is in the range

$$0 \leq \alpha \leq 1$$

as suggested from the data in Figure 3. We may also show that the location of the solution to Eq. (20) is not especially sensitive to the parameter setting. Therefore, it is possible to use any efficient iterative technique for solving Eq. (20) and we have found that such techniques converge quite rapidly to the optimal solution.

4. Comparison of solutions for various cost functions. In the last three subsections we have considered three different cost functions: the linear cost function; the logarithmic cost function; and the power law cost function. Of course we see immediately that the linear cost function is a special case $\alpha = 1$ of the power law cost function. We wish now to compare the performance and cost of computer networks under these various cost functions. We use for our example the ARPA computer network as described above.

It is not obvious how one should proceed in making this comparison. However, we adopt the following approach in an attempt to make some meaningful comparisons. We consider the ARPA network at a traffic load of 100% of the

full data rate, namely 225 kilobits per second (denoted by γ_0). For the 50 kilobit net shown in Figure 1 we may calculate the line costs from Table II (eliminating the termination charges since we recognize this causes no essential change in our optimization procedures, as mentioned above); the resultant network cost is approximately \$579,000 per year (which we denote by D_0). Using this γ_0 and D_0 (as well as the other given input parameters) we may then carry out the optimization indicated in Eq. (6) for the case of a linear cost function where $d_i = AL_i$ and A is immediately found from the mileage cost in Table II. This calculation results in an average message delay T_0 (calculated from Eq.(14)) whose value is approximately 24 milliseconds. We have now established an "operating point" for the three quantities γ_0, D_0, and T_0, whose values are 100% of full data rate, \$579,000, and 24 milliseconds, respectively.

We may now examine all of our other cost functions by forcing them to pass through this operating point. We assume $d_i = AL_i$ throughout for these calculations. Also we choose $\alpha = 1$ for the logarithmic case in Eq. (16). (Note for the logarithmic and power law cases, that two unknown constants, B and A, must be determined; this is now easily done if we set $T = T_0$ and $D = D_o$ for $\gamma = \gamma_0$ in each of these two cases independently.) In particular now we wish to examine the behavior of the network under these various cost functions. We do this first by fixing the cost of the network at $D = D_0$ and plotting T, the average time delay, as we vary the percentage of full data rate applied to the network; this performance is given in Figure 4 where we show the system behavior for the power law cost function and the linear cost function. The

result is striking! We see that the variation in average message delay is almost insignificant as α passes through the range from 0.3 to 1.0. We conclude then that the very important power law cost function may be analyzed using a linear cost function when one is interested in evaluating the average time delay at fixed cost.*

We also consider the variation of the network cost D as a function of data rate at fixed average message delay, namely $T = T_0 = 24$ milliseconds. This performance is shown in Figure 5 for all three cost functions. We note here that the linear cost function is only a fair approximation to the power law cost function over the range of α shown; the logarithmic cost function is also shown and behaves very much like the linear cost function for data rates above γ_0 but departs from that behavior for data rates below γ_0. It can be shown that the network cost, D, at fixed $T = T_0$ for the case $\alpha = 1$ (linear cost function) varies as a constant plus a linear dependence on γ. It is also of interest to cross plot the average time delay T with the network cost D. This we do in Figure 6 for the class of power law cost functions. In Figures 6a and 6b we obtain points along the vertical and horizontal axes corresponding to fixed delay and fixed cost, respectively. These loci are obtained by varying γ and we connect the points for equal γ with straight lines as shown in the figure (however, we in no way imply that the system passes along these straight lines as both T and D are allowed to vary simultaneously). We note the increased range of D as α varies from 0.3 to 1.0, but very little change in the range of

---

*The logarithm cost function is not shown in Figure 4 since the time delay is extremely sensitive to the data rate and bears little resemblance to the power law case.

T. In Figure 6c we collect together the behavior in this plane for many values of α where the lines labelled with a particular value of α correspond to the 50% data rate case in the lower left-hand portion of the figure and to the 130% data rate case in the upper right-hand portion of the figure. From Figure 6c we clearly observe that for fixed cost the time delay range varies insignificantly as α changes (as we emphasized in discussing Figure 4). Similarly, we observe the moderate variation at fixed time delay of network cost as α ranges through its values (this we saw clearly in Figure 5).

These studies of network optimization for various cost functions need further investigation. Our aim in this section has been to exhibit some of the performance characteristics under these cost functions and to compare them in some meaningful way.

Simulated Routing in the ARPA Network--Operating Procedure

We have examined analysis and synthesis procedures for computer networks above. We now proceed to exhibit some properties of the network operating procedure, in particular, the message routing procedure.

The ARPA network uses a routing procedure which is local in nature as opposed to global. Some details of this procedure are available in Reference 6 in these proceedings and we wish to comment on the method used for updating the routing tables. For purposes of routing, each node maintains a list which contains for each destination an estimate of the delay a message would encounter in attempting to reach that destination node were it to be sent out over a particular channel emanating from that node; the list contains an entry for each destination and each line leaving the node in which this list is contained. Every half second (approximately) each node sends to all of its immediate neighbors a list which contains its estimate of the shortest delay time to pass to

each destination; this list therefore contains a number of entries which is one less than the number of nodes in the network. Upon receiving this information from one of its neighbors the IMP adds to this list of estimated delays, a measure of the current delays in passing from itself to the neighbor from whom it is receiving this list; this then provides that IMP an estimate of the minimum delay required to reach all destinations if one travelled out over the line connected to that neighbor. The routing table for the IMP is then constructed by combining the lists of all of its neighbors into a set of columns and choosing as the output line for messages going to a particular destination that line for which the estimated delay over that line to that destination is minimum. What we have here described is essentially a periodic or synchronous updating method for the routing tables as currently used in the ARPA network. It has the clear advantages of providing reasonably accurate data regarding path delays as well as the important advantage of being a rather simple procedure both from an operational point of view and from an overhead point of view in terms of software costs inside the IMP program.

We suggest that a more efficient procedure in terms of routing delays is to allow asynchronous updating; by this we mean that routing information is passed from a node to its nearest neighbors only when significant enough changes occurred in its own routing table to warrant such an information exchange. The definition of "significant enough" must be studied carefully but certainly implies the use of thresholds on the percentage change of estimated delays. When these thresholds are crossed in an IMP then routing information is transferred to that IMP's nearest neighbors. This asynchronous mode of updating implies a large overhead for updating and it remains to be seen whether the

advantages gained through this more elaborate updating method overcome the disadvantages due to software costs and cycle-stealing costs for updating. We may observe the difference in performance between synchronous and asynchronous updating through the use of simulation as shown in Figure 7. In this figure we plot the average time delay T versus the average path length for messages under various routing disciplines. We observe immediately that the three points shown for asynchronous updating are significantly superior to those shown for synchronous updating. For a comparison we also show the result of a fixed routing algorithm which was computed by solving for the shortest delay path in an unloaded network; the asynchronous updating shows superior performance to the fixed routing procedure. However, the synchronous updating shows inferior performance compared to this very simple fixed routing procedure if we take as our performance measure the average message delay.

It was observed that with synchronous updating it was possible for a message to get trapped temporarily in loops (i.e., travelling back and forth between the same pair of nodes) We suppressed this looping behavior for two synchronous updating procedures with different parameter settings and achieved significant improvement; nevertheless, this improved version remains inferior to those simulated systems with asynchronous updating. As mentioned above, asynchronous updating contains many virtues, but one must consider the overhead incurred for such a sophisticated updating procedure before it can be incorporated and expected to yield a net improvement in performance.

CONCLUSIONS

Our goal in this paper has been to demonstrate the importance of analytical and simulation techniques in evaluating computer networks in the early design stages. We have addressed ourselves to three areas of interest, namely the analysis of computer network performance using methods from queueing theory, the optimal synthesis problem for a variety of cost functions, and the choice of routing procedure for these networks. Our results show that it is possible to obtain exceptionally good results in the analysis phase when one considers the "small" packet traffic only. As yet, we have not undertaken the study of the multi-packet traffic behavior. In examining available data we found that the power law cost function appears to be the appropriate one for high-speed data lines. We obtain optimal channel capacity assignment procedures for this cost function as well as the logarithmic cost function and the linear cost function. A significant result issued from this study through the observation that the average message delay for the power law cost function could very closely be approximated by the average message delay through the system constrained by a linear cost function; this holds true in the case when the system cost is held fixed. For the fixed delay case we found that the variation of the system cost under a power law constraint could be represented by the cost variation for a linear cost constraint only to a limited extent.

In conjunction with pure analytical results it is extremely useful to take advantage of system simulation. This is the approach we describe in studying the effect of routing procedures and comparing methods for updating these procedures. We indicated that asynchronous updating was clearly superior to synchronous updating except in the case where the overhead for asynchronous

updating might be severe.

The results referred to above serve to describe the behavior of computer network systems and are useful in the early stages of system design. If one is desirous of obtaining numerical tools for choosing the precise design parameters of a system, then it is necessary to go to much more elaborate analytic models or else to resort to efficient search procedures (such as that described in Reference 4) in order to locate optimal designs.

Acknowledgments

The author is pleased to acknowledge Gary L. Fultz for his assistance in simulation studies as well as his contributions to loop suppression in the routing procedures; acknowledgment is also due to Ken Chen for his assistance in the in/numerical solution for the performance under different cost function constraints.

REFERENCES

1 S CARR S CROCKER V CERF
Host to host communication protocol in the ARPA network
These proceedings

2 P A DICKSON
ARPA network will represent integration on a large scale
Electronics September 30 1968 131-134

3 R G GOULD
Comments on generalized cost expressions for private-line communications channels
IEEE Transactions on Communication Technology, V Com-13 No 3 September 1965 374-377

also

R P ECKHERT P N KELLY
A program for the development of a computer resource sharing network
Internal Report for Kelly Scientific Corp Washington D C
February 1969

4 H FRANK I T FRISCH W CHOU
Topological considerations in the design of the ARPA computer network
These proceedings

5 F B HILDEBRAND
Methods of Applied Mathematics
Prentice-Hall Inc Englewood Cliffs N J 1958

6 F E HEART R E KAHN S M ORNSTEIN W R CROWTHER D C WALDEN
The interface message processor for the ARPA network
These proceedings

7 L KLEINROCK
Communication nets; stochastic message flow and delay
McGraw-Hill New York 1964

8 L KLEINROCK
Models for computer networks
Proc of the International Communications Conference University of Colorado Boulder June 1969 21-9 to 21-16

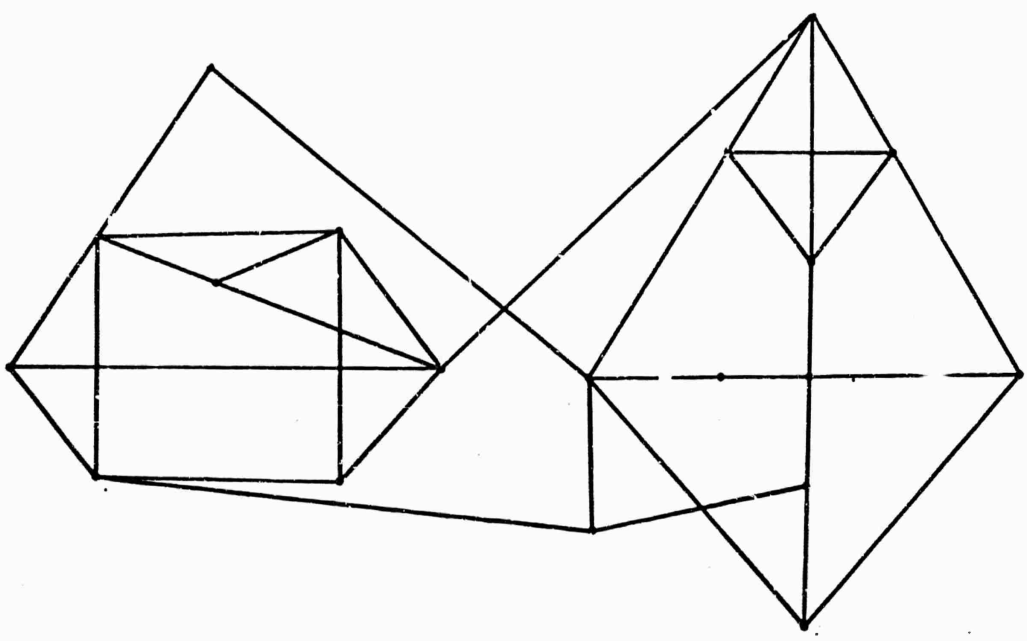
9 L KLEINROCK
Comparison of solutions methods for computer network models
Proc of the Computers and Communications Conference Rome New York September 30 - October 2 1969

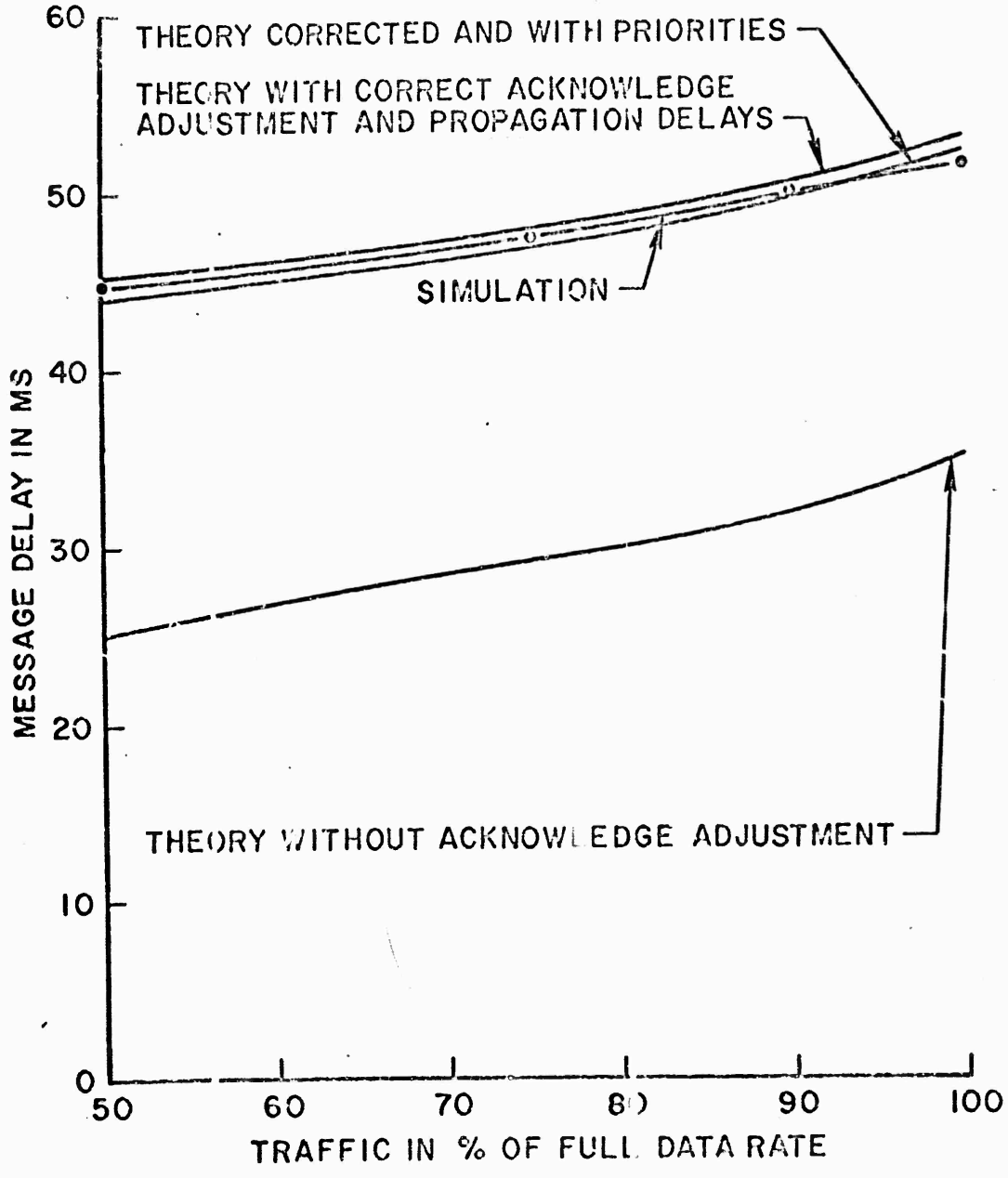
10 F R MASTRAMONACO
Optimum speed of service in the design of customer data communications systems
Proc of the ACM Symposium on the Optimization of Data Communications Systems Pine Mountain Georgia October 13-16 1969 127-151

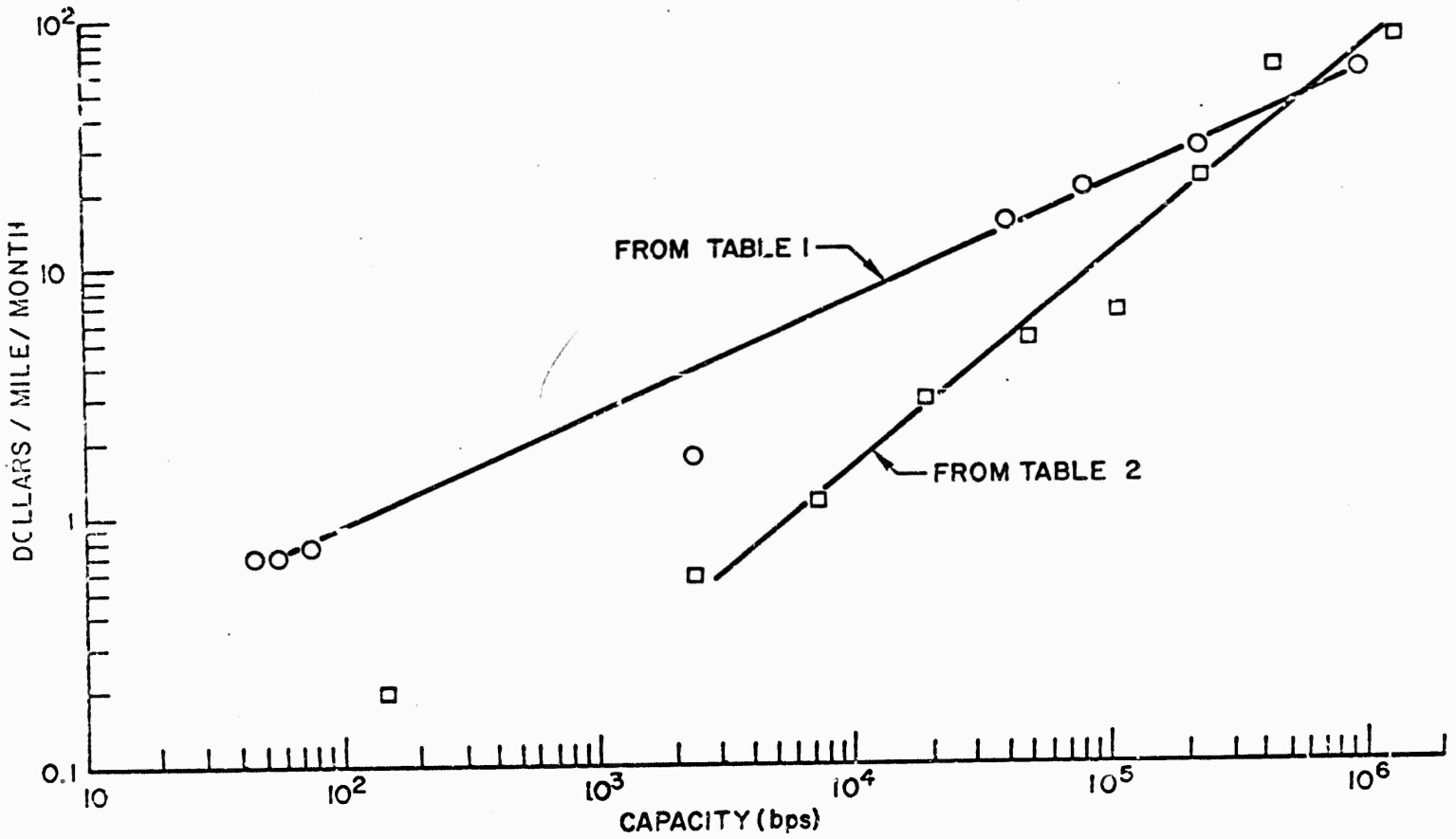
11 L G ROBERTS
Multiple computer networks and intercomputer communications
ACM Symposium on Operating Systems Principles Gatlinburg Tennessee October 1967

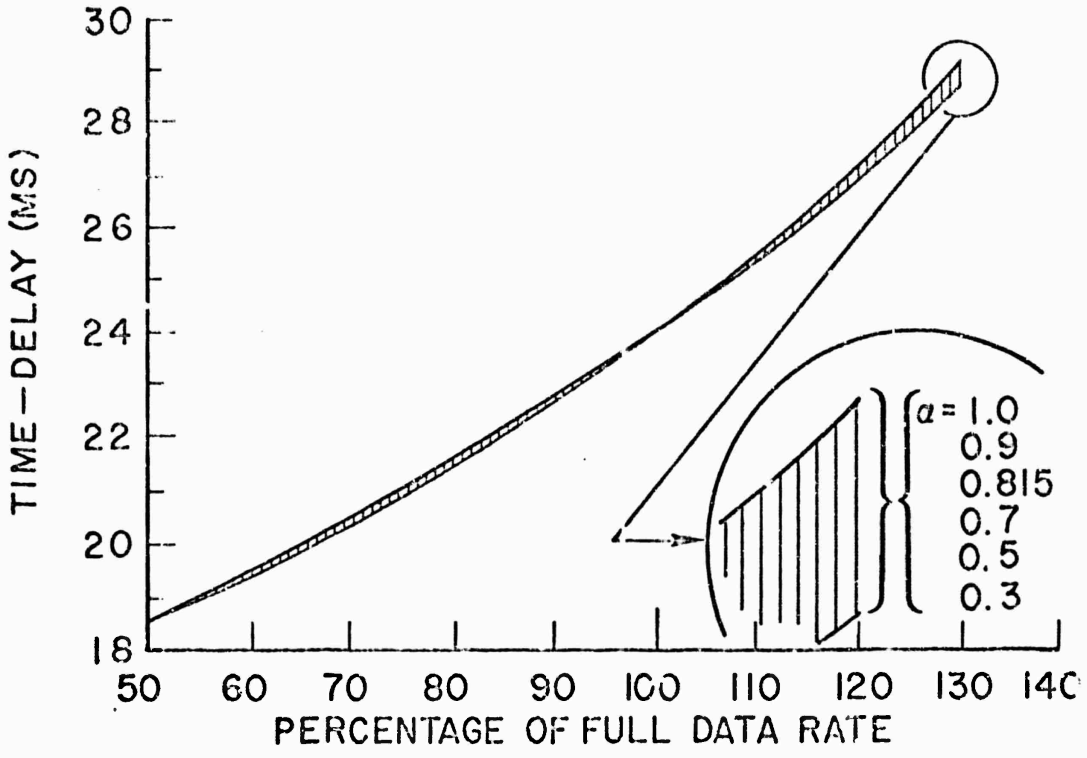
12 L G ROBERTS B D WESSLER
Computer network developments to achieve resource sharing
These proceedings

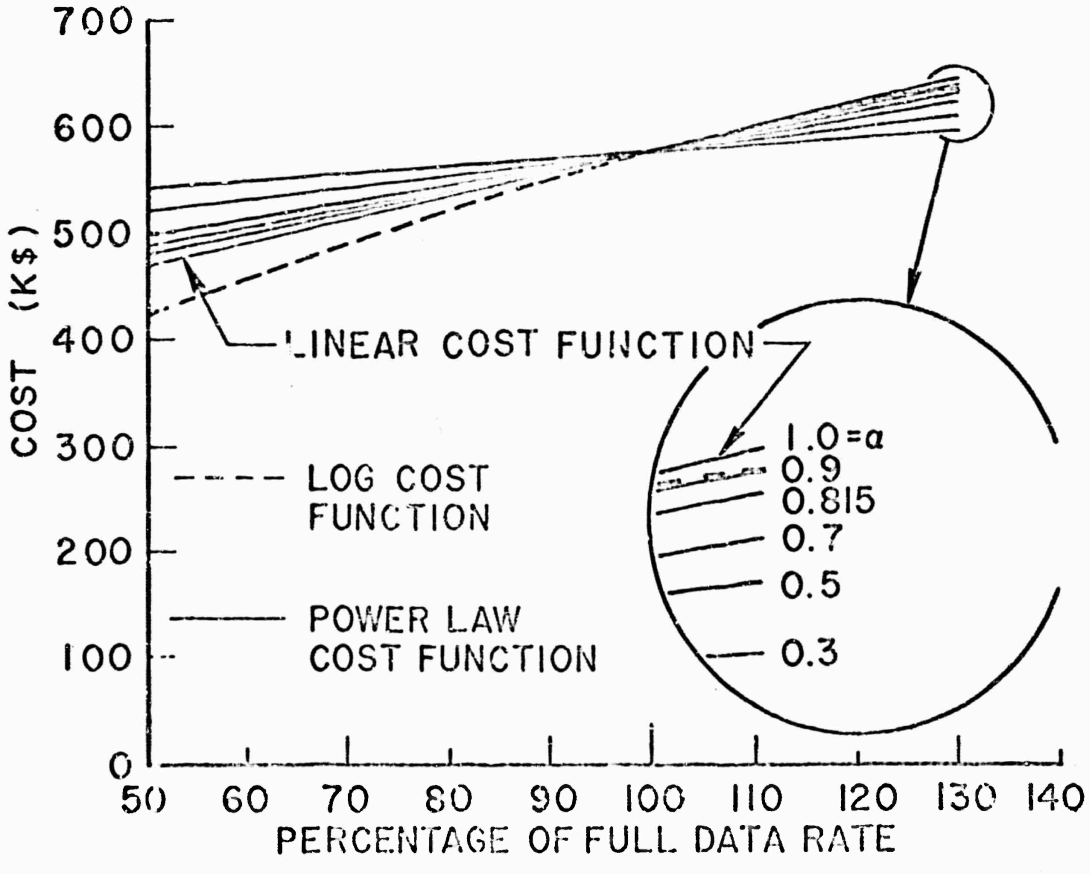
List of Figures

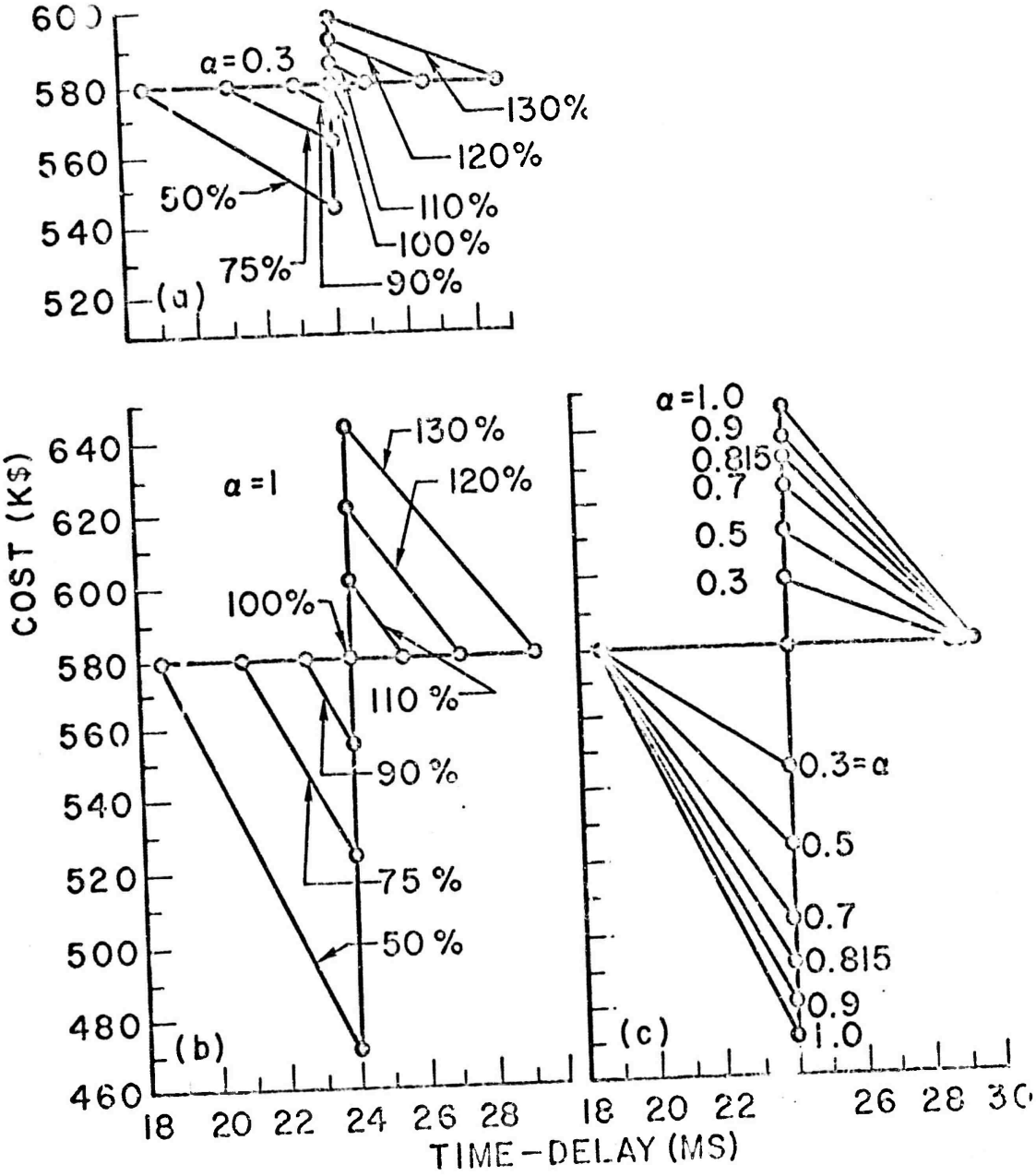


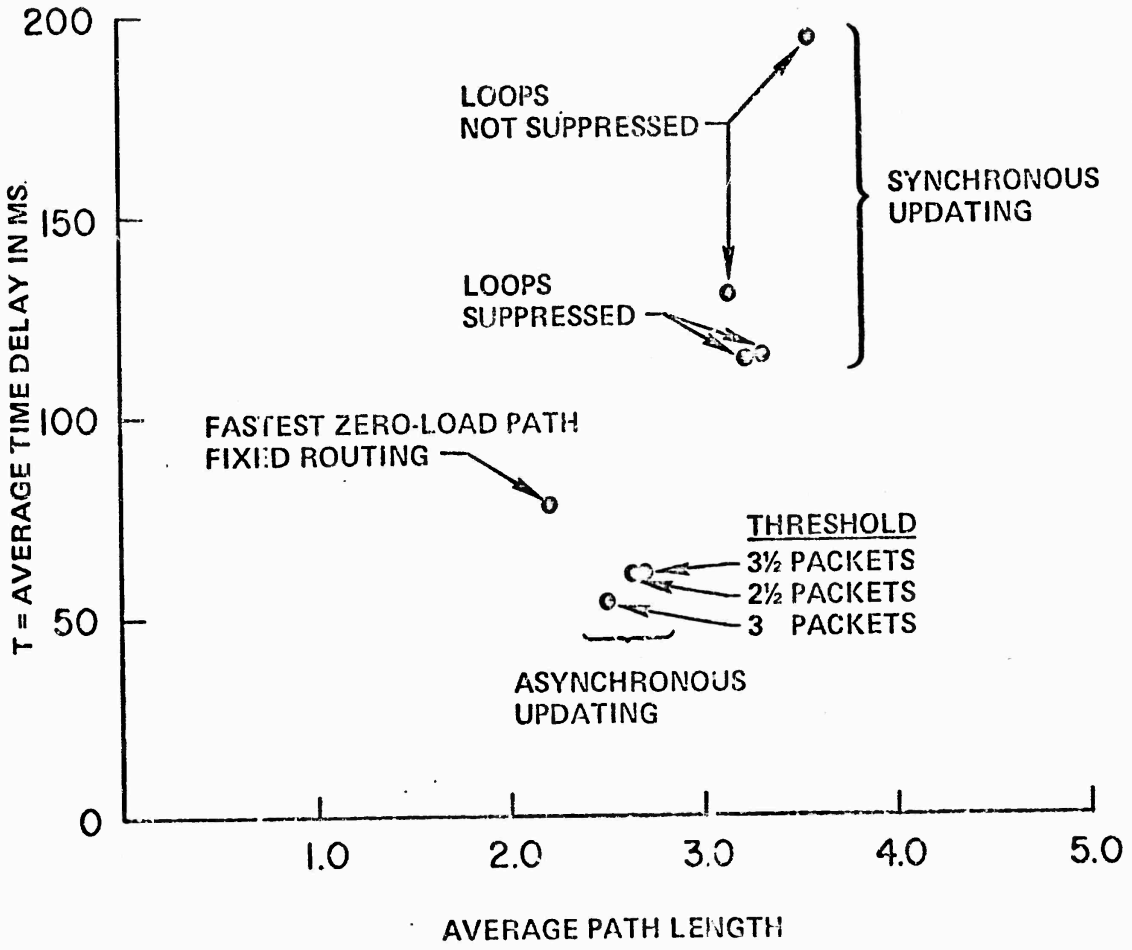

60
50
40
30
20
10
0
MESSAGE DELAY IN MS
THEORY CORRECTED AND WITH PRIORITIES
THEORY WITH CORRECT ACKNOWLEDGE ADJUSTMENT AND PROPAGATION DELAYS
SIMULATION
THEORY WITHOUT ACKNOWLEDGE ADJUSTMENT
50
60
70
80
90
100
TRAFFIC IN % OF FULL DATA RATE


10^2
10
1
0.1
DOLLARS / MILE / MONTH
FROM TABLE 1
FROM TABLE 2
10
10^2
10^3
10^4
10^5
10^6
CAPACITY (bps)


TIME–DELAY (MS)
30
28
26
24
22
20
18
50 60 70 80 90 100 110 120 130 140
PERCENTAGE OF FULL DATA RATE
α = 1.0
0.9
0.815
0.7
0.5
0.3


COST (K$)
700
600
500
400
300
200
100
0
50 60 70 80 90 100 110 120 130 140
PERCENTAGE OF FULL DATA RATE
LINEAR COST FUNCTION
LOG COST FUNCTION
POWER LAW COST FUNCTION
1.0=α
0.9
0.815
0.7
0.5
0.3


600
580
560
540
520
α=0.3
130%
120%
110%
100%
90%
75%
50%
(a)
COST (K$)
640
620
600
580
560
540
520
500
480
460
α=1
130%
120%
100%
110%
90%
75%
50%
(b)
α=1.0
0.9
0.815
0.7
0.5
0.3
0.3=α
0.5
0.7
0.815
0.9
1.0
(c)
18 20 22 24 26 28
18 20 22
26 28 30
TIME-DELAY (MS)


200
150
100
50
0
T = AVERAGE TIME DELAY IN MS
LOOPS
NOT SUPPRESSED
LOOPS
SUPPRESSED
SYNCHRONOUS
UPDATING
FASTEST ZERO-LOAD PATH
FIXED ROUTING
THRESHOLD
3½ PACKETS
2½ PACKETS
3 PACKETS
ASYNCHRONOUS
UPDATING
1.0
2.0
3.0
4.0
5.0
AVERAGE PATH LENGTH

II.2 HOST-HOST Communication Protocol in the ARPA Network*

(This is a paper to be presented at SJCC '70 written by

C. Stephen Carr, University of Utah, Salt Lake City, Utah

Stephen D. Crocker, University of California, Los Angeles, California

Vinton G. Cerf, University of California, Los Angeles, California)

*This research was sponsored by the Advanced Research Projects Agency, Department of Defense, under contracts AF30(602)-4277 and DAHC15-69-C-0285.

INTRODUCTION

The Advanced Research Projects Agency (ARPA) Computer Network (hereafter referred to as the "ARPA network") is one of the most ambitious computer networks attempted to date.[1] The types of machines and operating systems involved in the network vary widely. For example, the computers at the first four sites are an XDS 940 (Stanford Research Institute), an IBM 360/75 (University of California, Santa Barbara), an XDS SIGMA-7 (University of California, Los Angeles), and a DEC PDP-10 (University of Utah). The only commonality among the network membership is the use of highly interactive time-sharing systems; but, of course, these are all different in external appearance and implementation. Furthermore, no one node is in control of the network. This has insured generality and reliability but complicates the software.

Of the networks which have reached the operational phase and been reported in the literature, none have involved the variety of computers and operating systems found in the ARPA network. For example, the Carnegie-Mellon, Princeton, IBM network consists of 360/67's with identical software.[2] Load sharing among identical batch machines was commonplace at North American Rockwell Corporation in the early 1960's. Therefore, the implementers of the present network have been only slightly influenced by earlier network attempts.

However, early time-sharing studies at the University of California at Berkeley, MIT, Lincoln Laboratory, and System Development Corporation (all ARPA sponsored) have had considerable influence on the design of the network. In some sense, the ARPA network of time-shared computers is a natural extension of earlier time-sharing concepts.

The network is seen as a set of data entry and exit points into which individual computers insert messages destined for another (or the same) computer, and from which such messages emerge. The format of such messages and the operation of the network was specified by the network contractor (BB&N) and it became the responsibility of representatives of the various computer sites to impose such additional constraints and provide such protocol as necessary for users at one site to use resources at foreign sites. This paper details the decisions that have been made and the considerations behind these decisions.

Several people deserve acknowledgment in this effort. J. Rulifson and W. Duvall of SRI participated in the early design effort of the protocol and in the discussions of NIL. C. Deloche of Thomson-CSF participated in the design effort while he was at UCLA and provided considerable documentation. J. [illegible] of Utah and P. Rovner of Lincoln Laboratory reviewed the early design and NIL. W. Crowther of Bolt, Beranek and Newman contributed the idea of a virtual net. The BB&N staff provided substantial assistance and guidance while delivering the network.

We have found that, in the process of connecting machines and operating systems together, a great deal of rapport has been established between personnel at the various network node sites. The resulting mixture of ideas, discussions, disagreements, and resolutions has been highly refreshing and

beneficial to all involved, and we regard the human interaction as a valuable by-product of the main effort.

THE NETWORK AS SEEN BY THE HOSTS

Before going on to discuss operating system communication protocol, some definitions are needed.

A HOST is a computer system which is part of the network.

An IMP (Interface Message Processor) is a Honeywell DDP-516 computer which interfaces with up to four HOSTs at a particular site, and allows HOSTs access into the network. The configuration of the initial four-HOST network is given in Figure 1. The IMPs form a store-and-forward communications network. A companion paper in these proceedings covers the IMPs in some detail.[3]

A message is a bit stream less than 8096 bits long which is given to an IMP by a HOST for transmission to another HOST. The first 32 bits of the message are the leader. The leader contains the following information:

(a) HOST

(b) Message type

(c) Flags

(d) Link number

When a message is transmitted from a HOST to its IMP, the HOST field of the leader names the receiving HOST. When the message arrives at the receiving HOST, the HOST field names the sending HOST.

Only two message types are of concern in this paper. Regular messages are generated by a HOST and sent to its IMP for transmission to a foreign

HOST. The other message type of interest is a RFNM (Request-for-Next-Message). RFNM's are explained in conjunction with links.

The flag field of the leader controls special cases not of concern here.

The link number identifies over which of 256 logical paths (links) between the sending HOST and the receiving HOST the message will be sent. Each link is unidirectional and is controlled by the network so that no more than one message at a time may be sent over it. This control is implemented using RFNM messages. After a sending HOST has sent a message to a receiving HOST over a particular link, the sending HOST is prohibited from sending another message over that same link until the sending HOST receives a RFNM. The RFNM is generated by the IMP connected to the receiving HOST, and the RFNM is sent back to the sending HOST after the message has entered the receiving HOST It is important to remember that there are 256 links in each direction and that no relationship among these is imposed by the network.

The purpose of the link and RFNM mechanism is to prohibit individual users from overloading an IMP or a HOST. Implicit in this purpose is the assumption that a user does not use multiple links to achieve a wide band, and to a large extent the HOST-HOST protocol cooperates with this assumption. An even more basic assumption, of course, is that the network's load comes from some users transmitting sequences of messages rather than many users transmitting single messages coincidently.

In order to delimit the length of the message, and to make it easier for HOSTs of differing word lengths to communicate, the following formatting procedure is used. When a HOST prepares a message for output, it creates a 32-bit leader. Following the leader is a binary string, called marking, consisting of an arbitrary number of zeroes, followed by a one. Marking

makes it possible for the sending HOST to synchronize the beginning of the text of a message with its word boundaries. When the last bit of a message has entered an IMP, the hardware interface between the IMP and HOST appends a one followed by enough zeroes to make the message length a multiple of 16 bits. These appended bits are called padding. Except for the marking and padding, no limitations are placed on the text of a message. Figure 2 shows a typical message sent by a 24-bit machine

DESIGN CONCEPTS

The computers participating in the network are alike in two important respects: each supports research independent of the network, and each is under the discipline of a time-sharing system. These facts contributed to the following design philosophy.

First, because the computers in the network have independent purposes it is necessary to preserve decentralized administrative control of the various computers. Since all of the time-sharing supervisors possess elaborate and definite accounting and resource allocation mechanisms, we arranged matters so that these mechanisms would control the load due to the network in the same way they control locally generated load.

Second, because the computers are all operated under time-sharing disciplines, it seemed desirable to facilitate basic interactive mechanisms.

Third, because this network is used by experienced programmers it was imperative to provide the widest latitude in using the network. Restrictions concerning character sets, programming languages, etc. would not be tolerat

and we avoided such restrictions.

Fourth, again because the network is used by experienced programmers, it was felt necessary to leave the design open-ended. We expect that conventions will arise from time to time as experience is gained, but we felt constrained not to impose them arbitrarily.

Fifth, in order to make network participation comfortable, or in some cases, feasible, the software interface to the network should require minimal surgery on the HOST operating system.

Finally, we accepted the assumption stated above that network use consists of prolonged conversations instead of one-shot requests.

These considerations led to the notions of connections, a Network Control Program, a control link, control commands, sockets, and virtual nets.

A connection is an extension of a link. A connection connects two processes so that output from one process is input to the other. Connections are simplex, so two connections are needed if two processes are to converse in both directions.

Processes within a HOST communicate with the network through a Network Control Program (NCP). In most HOSTs, the NCP will be part of the executive, so that processes will use system calls to communicate with it. The primary function of the NCP is to establish connections, break connections, switch connections, and control flow.

In order to accomplish its tasks, a NCP in one HOST must communicate with a NCP in another HOST. To this end, a particular link between each pair of HOSTs has been designated as the control link. Messages received

over the control link are always interpreted by the NCP as a sequence of one or more control commands. As an example, one of the kinds of control commands is used to assign a link and initiate a connection, while another kind carries notification that a connection has been terminated. A partial sketch of the syntax and semantics of control commands is given in the next section.

A major issue is how to refer to processes in a foreign HOST. Each HOST has some internal naming scheme, but these various schemes often are incompatible. Since it is not practical to impose a common internal process naming scheme, an intermediate name space was created with a separate portion of the name space given to each HOST. It is left to each HOST to map internal process identifiers into its name space.

The elements of the name space are called sockets. A socket forms one end of a connection, and a connection is fully specified by a pair of sockets. A socket is specified by the concatenation of three numbers:

(a) a user number (24 bits)

(b) a HOST number (8 bits)

(c) AEN (8 bits)

A typical socket is illustrated in Figure 3.

Each HOST is assigned all sockets in the name space which have field (b) equal to the HOST's own identification.

A socket is either a receive socket or a send socket, and is so marked by the low-order bit of the AEN (0 = receive, 1 = send). The other seven bits of the AEN simply provide a sizable population of sockets for each user number at each HOST. (AEN stands for "another eight-bit number".)

Each user is assigned a 24-bit user number which uniquely identifies him throughout the network. Generally this will be the 8-bit HOST number of his home HOST, followed by 16 bits which uniquely identify him at that HOST. Provision can also be made for a user to have a user number not keyed to a particular HOST, an arrangement desirable for mobile users who might have no home HOST or more than one home HOST. This 24-bit user number is then used in the following manner. When a user signs onto a HOST, his user number is looked up. Thereafter, each process the user creates is tagged with his user number. When the user signs onto a foreign HOST via the network, his same user number is used to tag processes he creates in that HOST. The foreign HOST obtains the user number either by consulting a table at login time, as the home HOST does, or by noticing the identification of the caller. The effect of propagating the user's number is that each user creates his own virtual net consisting of processes he has created. This virtual net may span an arbitrary number of HOSTs. It will thus be easy for a user to connect his processes in arbitrary ways, while still permitting him to connect his processes with those in other virtual nets.

The relationship between sockets and processes is now describable (see Figure 4). For each user number at each HOST, there are 128 send sockets and 128 receive sockets. A process may request from the local NCP the use of any one of the sockets with the same user number; the request is granted if the socket is not otherwise in use. The key observation here is that a socket requested by a process cannot already be in use unless it is by some other process within the same virtual net, and

such a process is controlled by the same user.

An unusual aspect of the HOST-HOST protocol is that a process may switch its end of a connection from one socket to another. The new socket may be in any virtual net and at any HOST, and the process may initiate a switch either at the time the connection is being established, or later. The most general forms of switching entail quite complex implementation, and are not germane to the rest of this paper, so only a limited form will be explained. This limited form of switching provides only that a process may substitute one socket for another while establishing a connection. The new socket must have the same user number and HOST number, and the connection is still established to the same process. This form of switching is thus only a way of relabelling a socket, for no change in the routing of messages takes place. In the next section we document the system calls and control commands; in the section after next, we consider how login might be implemented.

SYSTEM CALLS AND CONTROL COMMANDS

Here we sketch the mechanics of establishing, switching and breaking a connection. As noted above, the NCP interacts with user processes via system calls and with other NCPs via control commands. We therefore begin with a partial description of system calls and control commands.

System calls will vary from one operating system to another, so the following description is only suggestive. We assume here that a process has several input-output paths which we will call ports. Each port may be

connected to a sequential I/O device, and while connected, transmits information in only one direction. We further assume that the process is blocked (dismissed, slept) while transmission proceeds. The following is the list of system calls:

Init <port>, <AEN 1>, <AEN 2>, <foreign socket>

where <port> is part of the process issuing the Init

and <AEN 1>, <AEN 2> are 8-bit AEN's (see Figure 3)

The first AEN is used to initiate the connection; the second is used while the connection exists.

<foreign socket> is the 40-bit socket name of the distant end of the connection.

The low-order bits of <AEN 1> and <AEN 2> must agree, and these must be the complement of the low-order bit of <foreign socket>.

The NCP concatenates <AEN 1> and <AEN 2> each with the user number of the process and the HOST number to form 40-bit sockets. It then sends a Request for Connection (RFC) control command to the distant NCP. When the distant NCP responds positively, the connection is established and the process is unblocked. If the distant NCP responds negatively, the local NCP unblocks the requesting process, but informs it that the system call has failed.

Listen <port>, <AEN 1>

where <port> and <AEN 1> are as above. The NCP retains the ports and <AEN 1> and blocks the process. When an RFC control command arrives naming the local socket, the process is unblocked and notified that a foreign process is calling.

Accept <AEN 2>

After a Listen has been satisfied, the process may either refuse the call or accept it and switch it to another socket. To accept the call, the process issues the Accept system call. The NCP then sends back an RFC control command.

Close <port>

After establishing a connection, a process issues a Close to break the connection. The Close is also issued after a Listen to refuse a call.

Transmit <port>, <addr>

If <port> is attached to a send socket, <addr> points to a message to be sent. This message is preceded by its length in bits.

If <port> is attached to a receive socket, a message is stored at <addr>. The length of the message is stored first.

Control commands

A vocabulary of control commands has been defined for communication between Network Control Programs. Each control command consists of an 8-bit operation code to indicate its function, followed by some parameters. The number and format of parameters is fixed for each operation code. A sequence of control commands destined for a particular HOST can be packed into a single control message.

RFC <my socket 1>, <my socket 2>,
<your socket>, (<link>)

This command is sent because a process has executed either an Init system call or an Accept system call. A link is assigned by the prospective receiver, so it is omitted if <my socket 1> is a send socket.

There is distinct advantage in using the same commands both to initiate a connection (Init) and to accept a call (Accept). If the responding command were different from the initiating command, then two processes could call each other and become blocked waiting for each other to respond. With this scheme no deadlock occurs and it provides a more compact way to connect a set of processes.

CLS <my socket>, <your socket>

The specified connection is terminated

CEASE <link>

When the receiving process does not consume its input as fast as it arrives, the buffer space in the receiving HOST is used to queue the waiting messages. Since only limited space is generally available, the receiving HOST may need to inhibit the sending HOST from sending any more

messages over the offending connection. When the sending HOST receives this command, it may block the process generating the messages.

RESUME <link>

This command is also sent from the receiving HOST to the sending HOST and negates a previous CEASE.

LOGGING IN

We assume that within each HOST there is always a process in execution which listens to login requests. We call this process the logger, and it is part of a special virtual net whose user number is zero. The logger is programmed to listen to calls on socket number 0. Upon receiving a call, the logger switches it to a higher (even) numbered sockets, and returns a call to the socket numbered one less than the send socket originally calling. In this fashion, the logger can initiate 127 conversations.

To illustrate, assume a user whose identification is X'010005' (user number 5 at UCLA) signs into UCLA, starts up one of his programs, and this program wants to start a process at SRI. No process at SRI except the logger is currently willing to listen to our user, so he executes

Init, <port> = 1, <AEN 1> = 7, <AEN 2> = 7,
<foreign socket> = 0.

His process is blocked, and the NCP at UCLA sends

RFC <my socket 1> = X'0100050107',
<my socket 2> = X'0100050107',
<your socket> = X'0000000200'

The logger at SRI is notified when this message is received, because it has previously executed

Lister <port> = 9, <AEN 1> = 0.

The logger then executes

Accept <AEN 2> = 88.

In response to the Accept, the SRI NCP sends

RFC <my socket 1> = X'0000000200'

<my socket 2> = X'0000000258'

<your socket> = X'0100050107'

<link> = 37

where the link has been chosen from the set of available links. The SRI logger then executes

Init <port> = 10

<AEN 1> = 89, <AEN 2> = 89,

<foreign socket> = X'0100050106'

which causes the NCP to send

RFC <my socket 1> = X'0000000259'

<my socket 2> = x'0000000259'

<your socket> = X'0100050106'

The process at UCLA is unblocked and notified of the successful Init. Because the SRI logger always initiates a connection to the AEN one less than it has just been connected to, the UCLA process then executes

Listen <port> = 11

<AEN 1> = 6

and when unblocked,

Accept <AEN 2> = 6.

When these transactions are complete, the UCLA process is doubly connected to the logger at SRI. The logger will then interrogate the UCLA process, and if satisfied, create a new process at SRI. This new process will be tagged with the user number X'010005', and both connections will be switched to the new process. In this case, switching the connections to the new process corresponds to "passing the console down" in many time-sharing systems.

USER LEVEL SOFTWARE

At the user level, subroutines which manage data buffers and format input destined for other HOSTs are provided. It is not mandatory that the user use such subroutines, since the user has access to the network system calls in his monitor.

In addition to user programming access, it is desirable to have a subsystem program at each HOST which makes the network immediately accessible from a teletype-like device without special programming. Subsystems are commonly used system components such as text editors, compilers and interpreters. An example of a network-related subsystem is TELNET, which will allow users at the University of Utah to connect to Stanford Research Institute and appear as regular terminal users. It is expected that more sophisticated subsystems will be developed in time, but this basic one will render the early network immediately useful.

A user at the University of Utah (UTAH) is sitting at a teletype

dialed into the University's PDP-10/50 time-sharing system. He wishes to operate the Conversational Algebraic Language (CAL) subsystem on the XDS-940 at Stanford Research Institute (SRI) in Menlo Park, California. A typical TELNET dialog is illustrated in Figure 5. The meaning of each line of dialog is discussed here.

(i) The user signs in at UTAH

(ii) The PDP-10 run command starts up the TELNET subsystem at the user's HOST.

(iii) The user identifies a break character which causes any message following the break to be interpreted locally rather than being sent on to the foreign HOST.

(iv) The TELNET subsystem will make the appropriate system calls to establish a pair of connections to the SRI logger. The connections will be established only if SRI accepts another foreign user.

The UTAH user is now in the pre-logged-in state at SRI. This is analogous to the standard teletype user's state after dialing into a computer and making a connection but before typing anything.

(v) The user signs in to SRI with a standard login command.

Characters typed on the user's teletype are transmitted unaltered through the PDP-10 (user HOST) and on to the 940 (serving HOST). The PDP-10 TELNET subsystem will have automatically switched to full-duplex, character-by-character transmission, since this is required by SRI's 940. Full duplex operation is allowed for by the PDP-10, though not used by most Digital Equipment Corporation subsystems.

(vi) and (vii) The 940 subsystem, CAL, is started.

At this point, the user wishes to load a CAL file into the 940 CAL subsystem from the file system on his local PDP-10.

(viii) CAL is instructed to establish a connection to UTAH in order to receive the file. "NETWRK" is a predefined 940 name similar in nature to "PAPER TAPE" or "TELETYPE".

(ix) Finally, the user types the break character (#) followed by a command to his PDP-10 TELNET program, which sends the desired file to SRI from Utah on the connection just established for this purpose. The user's next statement is in CAL again.

The TELNET subsystem coding should be minimal for it is essentially a shell program built over the network system calls. It effectively established a shunt in the user HOST between the remote user and a distant serving HOST.

Given the basic system primitives, the TELNET subsystem at the user HOST and a manual for the serving HOST, the network can be profitably employed by remote users today.

HIGHER LEVEL PROTOCOL

The network poses special problems where a high degree of interaction is required between the user and a particular subsystem on a foreign HOST. These problems arise due to heterogeneous consoles, local operating system overhead, and network transmission delays. Unless we use special strategies it may be difficult or even impossible for a distant

user to make use of the more sophisticated subsystems offered. While these difficulties are especially severe in the area of graphics, problems may arise even for teletype interaction. For example, suppose that a foreign subsystem is designed for teletype consoles connected by telephone, and then this subsystem becomes available to network users. This subsystem might have the following characteristics.

1. Except for echoing and correction of mistyping, no action is taken until a carriage return in typed.
2. All characters except "↑", "←" and carriage return are echoed as the character typed.
3. ← causes deletion of the immediately preceding accepted character, and is echosed as that character.
4. ↑ causes all previously typed characters to be ignored. A carriage return and line feed are echoed.
5. A carriage return is echoed as a carriage return followed by a line feed.

If each character typed is sent in its own message, then the characters

H E L L O ← ← P c.r.

cause nine messages in each direction. Furthermore, each character is handled by a user level program in the local HOST before being sent to the foreign HOST.

Now it is clear that if this particular example were important, we would quickly implement rules 1 to 5 in a local HOST program and send only complete lines to the foreign HOST. If the foreign HOST program could not be modified so as to not generate echoes, then the local program

could not only echo properly, it could also throw away the later echoes from the foreign HOST. However, the problem is not any particular interaction scheme; the problem is that we expect many of these kinds of schemes to occur. We have not found any general solutions to these problems, but some observations and conjectures may lead the way.

With respect to heterogeneous consoles, we note that although consoles are rarely compatible, many are equivalent. It is probably reasonable to treat a model 37 teletype as the equivalent of an IBM 2741. Similarly, most storage scopes will form an equivalence class, and most refresh display scopes will form another. Furthermore, a hierarchy might emerge with members of one class usable in place of those in another, but not vice versa. We can imagine that any scope might be an adequate substitute for a teletype, but hardly the reverse. This observation leads us to wonder if a network-wide language for consoles might be possible. Such a language would provide for distinct treatment of different classes of consoles, with semantics appropriate to each class. Each site could then write interface programs for its consoles to make them look like network standard devices.

Another observation is that a user evaluates an interactive system by comparing the speed of the system's responses with his own expectations. Sometimes a user feels that he has made only a minor request, so the response should be immediate; at other times he feels he has made a substantial request, and is therefore willing to wait for the response. Some interactive subsystems are especially pleasant to use because a great deal of work has gone into tailoring the responses to the user's

expectations. In the network, however, a local user level process intervenes between a local console and a foreign subsystem, and we may expect the response time for minor requests to degrade. Now it may happen that all of this tailoring of the interaction is fairly independent of the portion of the subsystem which does the heavy computing or I/O. In such a case, it may be possible to separate a subsystem into two sections. One section would be the "substantive" portion; the other would be a "front end" which formats output to the user, accepts his inputs, and controls computationally simple responses such as echoes. In the example above, the program to accumulate a line and generate echoes would be the front end of some subsystem. We now take notice of the fact that the local HOSTs have substantial computational power, but our current designs make use of the local HOST only as a data concentrator. This is somewhat ironic, for the local HOST is not only poorly utilized as a data concentrator, it also degrades performance because of the delays it introduces.

These arguments have led us to consider the possibility of a Network Interface Language (NIL) which would be a network-wide language for writing the front end of interactive subsystems. This language would have the feature that subprograms communicate through network-like connections. The strategy is then to transport the source code for the front end of a subsystem to the local HOST, where it would be compiled and executed.

During preliminary discussions we have agreed that NIL should have at least the following semantic properties not generally found in languages.

1. Concurrency. Because messages arrive asynchronously on different connections, and because user input is not synchronized with subsystem output, NIL must include semantics to accurately model the possible concurrencies.
2. Program Concatenation. It is very useful to be able to insert a program in between two other programs. To achieve this, the interconnection of programs would be specified at run time and would not be implicit in the source code.
3. Device substitutability. It is usual to define languages so that one device may be substituted for another. The requirement here is that any device can be modelled by a NIL program. For example, if a network standard display controller manipulates tree-structures according to messages sent to it then these structures must be easily implementable in NIL.

NIL has not been fully specified, and reservations have been expressed about its usefulness. These reservations hinge upon our conjecture that it is possible to divide an interactive subsystem into a transportable front end which satisfies a user's expectations at low cost and a more substantial stay-at-home section. If our conjecture is false, then NIL will not be useful; otherwise it seems worth pursuing. Testing of this conjecture and further development of NIL will take priority after low level HOST-HOST protocol has stabilized.

HOST/IMP INTERFACING

The hardware and software interfaces between HOST and IMP is an area of particular concern to the HOST organizations. Considering the diversity of HOST computers to which a standard IMP must connect, the hardware interface was made bit serial and full-duplex. Each HOST organization implements its half of this very simple interface.

The software interface is equally simple and consists of messages passed back and forth between the IMP and HOST programs. Special error and signal messages are defined as well as messages containing normal data. Messages waiting in queues in either machine are sent at the pleasure of the machine in which they reside with no concern for the needs of the other computer.

The effect of the present software interface is the needless re-buffering of all messages in the HOST in addition to the buffering in the IMP. The messages have no particular order other than arrival times at the IMP. The Network Control Program at one HOST (e.g., Utah) needs waiting RFNM's before all other messages. At another site (e.g., SRI), the NCP could benefit by receiving messages for the user who is next to be run.

What is needed is coding representing the specific needs of the HOST on both sides of the interface to make intelligent decisions about what to transmit next over the channel. With the present software interface the channel in one direction once committed to a particular message is then locked up for up to 80 milliseconds! This approaches one teletype character time and needlessly limits full-duplex, character by character,

interaction over the net. At the very least, the IMP/HOST protocol should be expanded to permit each side to assist the other in scheduling messages over the channels.

CONCLUSIONS

At this time (February 1970) the initial network of four sites is just beginning to beginning to be utilized. The communications system of four IMPs and wide band telephone lines have been operational for two months. Programmers at UCLA have signed in as users of the SRI 940. More significantly, one of the authors (S. Carr) living in Palo Alto uses the Salt Lake PDP-10 on a daily basis by first connecting to SRI. We thus have first hand experience that remote interaction is possible and is highly effective.

Work on the ARPA network has generated new areas of interest. NIL is one example, and interprocess communication is another. Interprocess communication over the network is a subcase of general interprocess communication in a multiprogrammed environment. The mechanism of connections seems to be new, and we wonder whether this mechanism is useful even when the processes are within the same computer.

REFERENCES

1 L ROBERTS

The ARPA network

Invitational Workshop on Networks of Computers Proceedings

National Security Agency 1968 p 115 ff

2 R M RUTLEDGE et al

An interactive network of time-sharing computers

Proceedings of the 24th National Conference

Association for Computing Machinery 1969 p 431 ff

3 F E HEART R E KAHN S M ORNSTEIN W R CROWTHER D C WALDEN

The interface message processors for the ARPA network

These Proceedings

LIST OF FIGURES

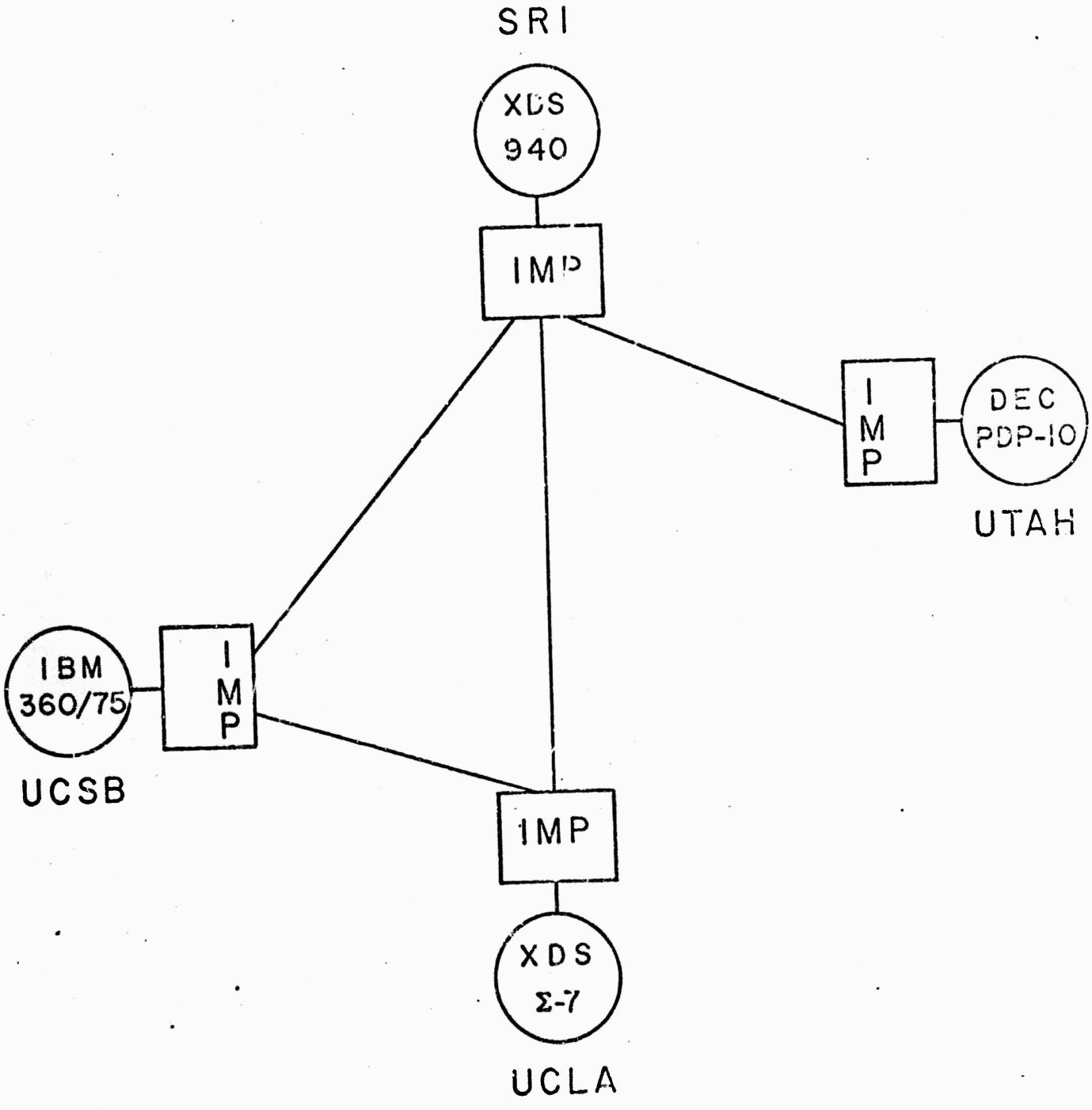


Figure 1 Initial network configuration

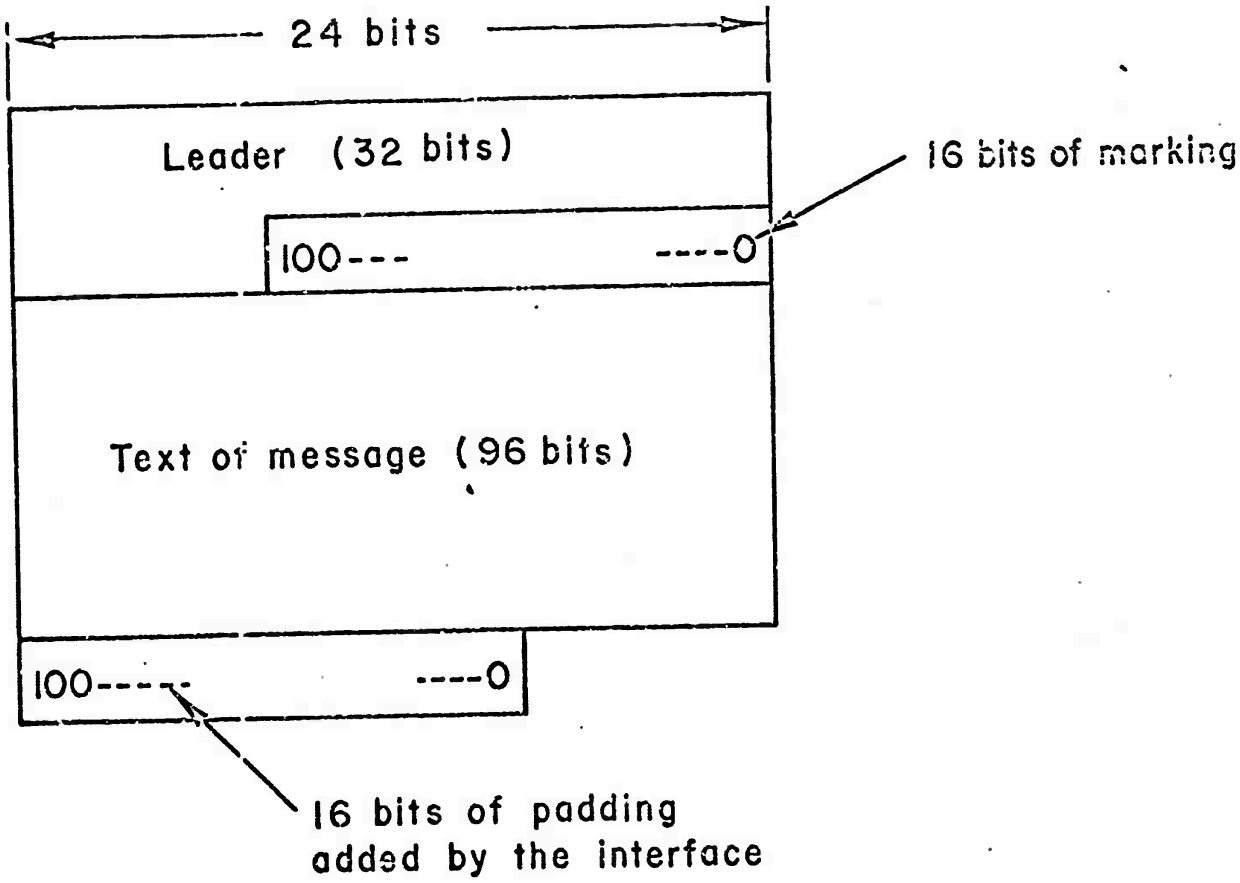


Figure 2 A typical message from a 24-bit machine

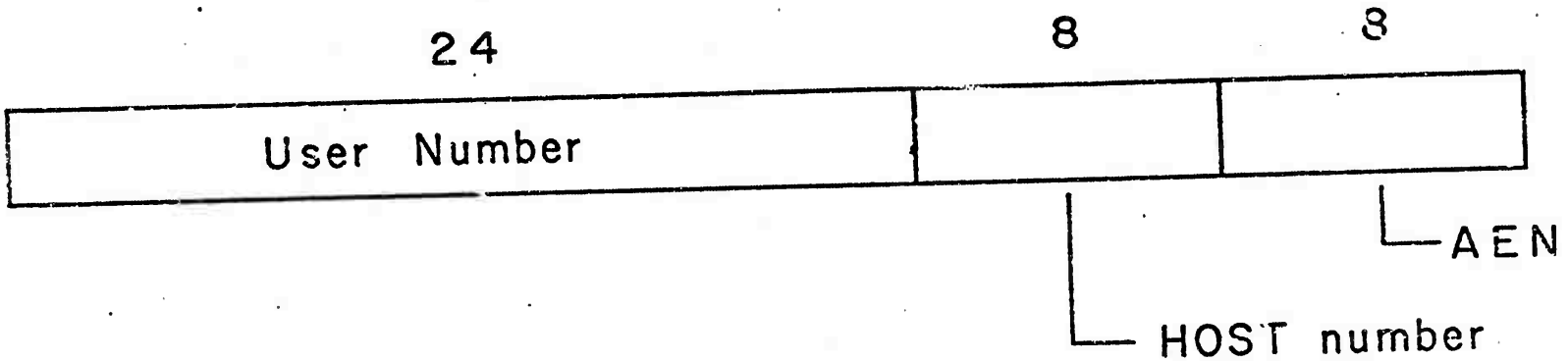


Figure 3 A typical socket

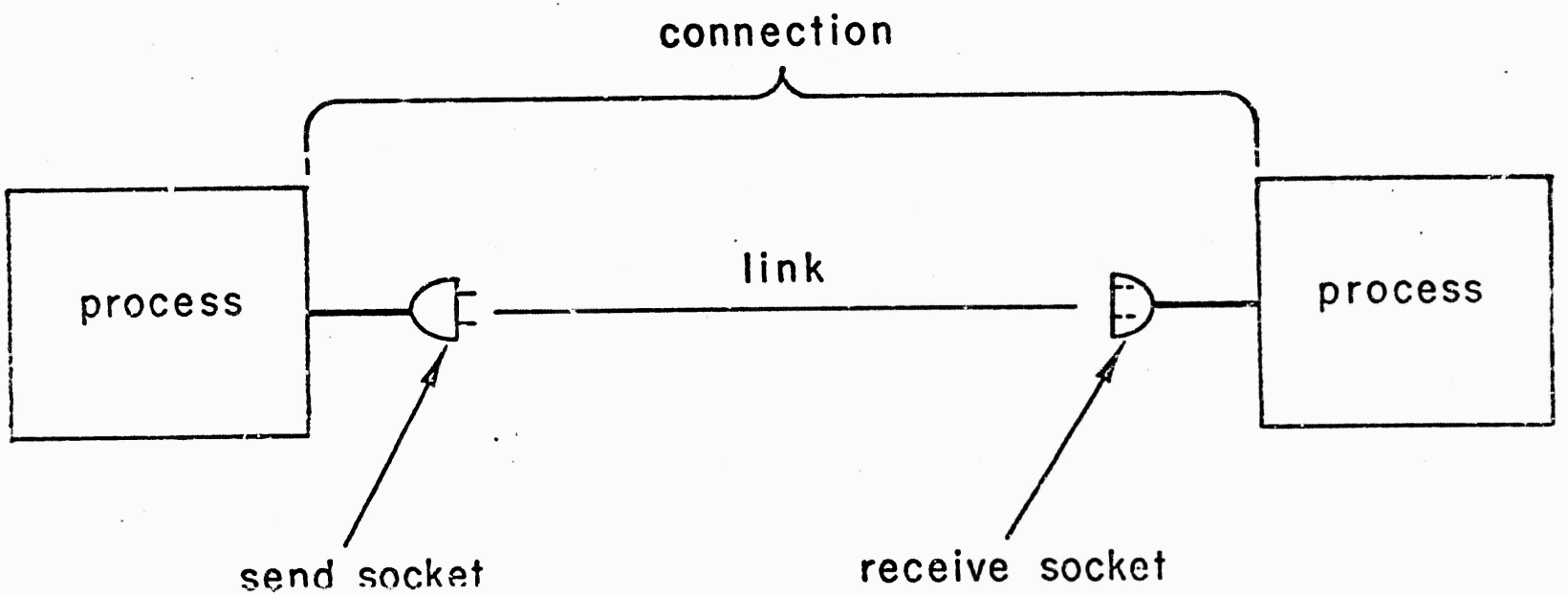


Figure 4 The relationship between sockets and processes

(i) .LOGIN(cr)

(ii) .R TELNET(cr)

(iii) ESCAPE CHARACTER IS #(cr)

(iv) CONNECT TO SRI(cr)

(v) @ENTER CARR.(cr)

(vi) @CAL.(cr)

(vii) CAL AT YOUR SERVICE(cr)

(viii) >READ FILE FROM NETWRK.(cr)

(ix) #NETWRK:←DSK:MYFILE.CAL(cr)

Figure 5 A typical TELNET dialog
Underlined characters are those typed by the user